МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КОЛЕСОВ ОЛЕКСАНДР ОЛЕКСАНДРОВИЧ

Допускається до захисту:
Завідувач кафедри
інформаційних технологій,
д.т.н., доцент,
_____Нескородєва Т. В.
«__»_____20__р.

РЕКОМЕНДАЦІЇ ІЗ ТЕСТУВАННЯ БЕЗПЕКИ API СЕРВІСІВ

Спеціальність 125 Кібербезпека

Кваліфікаційна (бакалаврська) робота

Науковий керівник:
Барибін О. І.,
к. т. н., доцент кафедри
інформаційних технологій

_____
(підпис)

Оцінка : _____ / _____ / _____
(бали/за шкалою ЄКТС/за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця 2022

АНОТАЦІЯ

Колесов О. О. Рекомендації із тестування безпеки API сервісів. Спеціальність 125 Кібербезпека. Донецький національний університет імені Василя Стуса. Вінниця. 2022 рік.

У бакалаврській роботі запропоновано набір інструментів як з ручного, так і автоматизованого тестування безпеки API сервісів, зокрема REST API, а також запропоновано методологію тестування у вигляді методичних рекомендацій на основі існуючих фреймворків оцінки безпеки інформаційних систем – ISSAF та OWASP Web Security Testing Guide. Згідно створених рекомендацій, процес тестування можна поділити на три послідовних фази: фаза збору інформації, фаза тестування механізмів безпеки API, а також фаза аналізу та документування результатів і очищення наслідків.

Ключові слова: рекомендації по тестуванню безпеки, API, веб API, REST API, інструменти.

Рис.: 15, Бібліограф.: 23 найм.

Kolesov O. O. Recommendations for testing the security of API services. Specialty 125 Cybersecurity. Vasyl Stus Donetsk National University. Vinnytsia. 2022.

The bachelor's thesis offers a set of tools for both manual and automated security testing of API services, including REST API, as well as testing methodology in the form of guidelines based on existing frameworks for assessing the security of information systems – ISSAF and OWASP Web Security Testing Guide. According to the recommendations, the testing process can be divided into three successive phases: the phase of collecting information, the phase of API security mechanisms testing, and the phase of analyzing and documenting results and cleaning the consequences.

Keywords: security testing guidelines, API, web API, REST API, tools.

Fig.: 15, Bibliographer: 23 items

ЗМІСТ

ВСТУП

У сучасному цифровому ландшафті багато повсякденної діяльності залежить від Інтернету. Він забезпечує спілкування, розваги, фінансові операції, отримання інформації у навчальних цілях та ін. Це означає, що тони даних і конфіденційної інформації постійно передаються через Інтернет.

Безпека в Інтернеті складається з низки заходів безпеки для захисту діяльності та транзакцій, що здійснюються онлайн через Інтернет. Ці заходи призначені для захисту користувачів від таких загроз, як злом комп'ютерних систем, адресів електронної пошти або веб-сайтів; шкідливе програмне забезпечення, яке може заражати і пошкоджувати системи; крадіжка особистих даних, таких як дані банківського рахунку та номери кредитних карток.

Усі клієнт-серверні взаємодії, за допомогою яких здійснюється обмін інформацією в мережі Інтернет, реалізуються за допомогою деякої абстракції – API.

Щоразу, коли користувач відвідує будь-яку сторінку в мережі, він взаємодіє з API віддаленого сервера. В даному випадку, API — це складова частина сервера, яка отримує запити та надсилає відповіді, тому можна сказати, що без такого ключового елементу, як API, існування сучасної всесвітньої веб мережі в сучасному її представленні неможливе. Відповідно, справедливо сказати, що рівень безпеки API серверної частини визначає рівень безпеки усього веб-ресурсу, тому його різностороннє тестування є **актуальним** як ніколи.

Метою роботи є формування структурованого та універсального ряду заходів по тестуванню заходів безпеки API сервісів у вигляді методичних рекомендацій.

Завданням роботи є:

- Дослідження поняття API, веб API та REST API як найбільш розповсюдженого представника веб API, аналізу основних загроз та вразливостей, а також механізмів забезпечення безпеки API сервісів.
- Аналіз існуючого набору інструментів та сфери їх застосування у тестуванні заходів безпеки API, наведення практичних прикладів його використання.
- Формування методичних рекомендацій по проведенню тестування безпеки API на основі існуючих методологій тестування безпеки інформаційних систем.

Об’єктом роботи є інформаційна безпека API сервісів.

Предметом дослідження є методичні рекомендації та інструментарій з тестування API сервісів.

1. РОЗДІЛ 1. ЗАГАЛЬНА ХАРАКТЕРИСТИКА API СЕРВІСІВ

1.1. Поняття API та його класифікація

API (акронім для інтерфейсу прикладного програмування, англ. Application Programming Interface) – це зв'язок між комп'ютерами або між комп'ютерними програмами, тип програмного інтерфейсу, який пропонує послуги іншим частинам програмного забезпечення. [1]

При створенні додатків, API значно спрощує програмування, абстрагуючи та інкапсулюючи основну реалізацію та показуючи лише об'єкти чи дії, необхідні розробнику та які будуть зрозумілими та інтуїтивними для кінцевого користувача. Для прикладу, графічний інтерфейс клієнта сервісу електронної пошти надає користувачеві кнопку, яка виконує всі кроки для отримання та виділення нових електронних листів. В свою чергу, API для введення/виводу файлів може надати розробнику функцію, яка копіює файл з одного місця в інше, що не вимагає від розробника розуміння того, як відбуваються операції з файловою системою, що виконуються «за кадром». Це дозволяє зменшити витрати ресурсів та часу, необхідних для розробки додатку.

Мільярди людей володіють смартфонами і використовують їх для обміну фотографіями в соціальних мережах. Це було б неможливо без API. Наприклад, обмін фотографіями за допомогою мобільної програми соціальної мережі передбачає використання різних типів API, як показано на рисунку 1.1.

Спершу, щоб зробити фото, мобільний додаток соціальної мережі використовує камеру смартфона через її API. Потім, знову ж за допомогою API камери, він може використовувати бібліотеку зображень, вбудовану в програму, щоб покращити якість фотографії (наприклад, збільшивши або зменшивши інтенсивність зображення). І, зрештою, він ділиться зміненою

фотографією, надсилаючи її на сервер соціальної мережі, використовуючи віддалений API, доступний через мережу Інтернет.

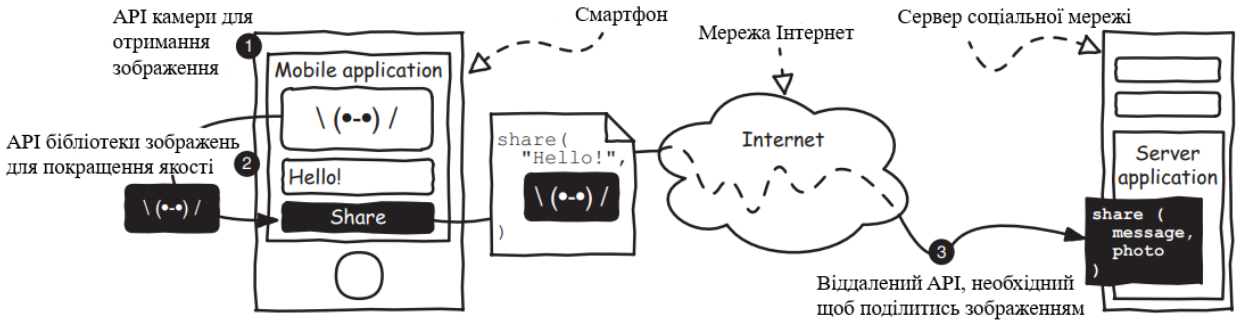


Рис. 1.1. Приклад роботи API сервісів різних типів [2]

Існує декілька класифікацій API сервісів. З наведеного вище прикладу можна виділити першу із специфікацій – класифікація по їх призначенню. [2] У результаті, в прикладі задіяні три різні типи API: відповідно, апаратний API (API камери), бібліотечний API та віддалений API.

Також існує класифікація API сервісів по їх доступності. Вирізняють два типи API: приватні та публічні. [3]

Публічні API – API, що відкриті для сторонніх девелоперів. Додатки з відкритими API здебільшого спрямовані на кінцевих користувачів. Їх ціллю є збільшення рівню інформованості про бренд та стимулювання зовнішніх інновацій.

Приватні API – це API з обмеженим доступом. Такі API також можна поділити на партнерські та внутрішні. Партнерські приватні API можуть відкрито рекламуватись, але надаються лише діловим партнерам, які підписали угоду з видавцем. Загальним випадком використання партнерських API є інтеграція програмного забезпечення між двома сторонами.

Відповідно, внутрішні API застосовуються лише внутрішнім персоналом на проектах організації, якій належать API. Девелопери або підрядники організації можуть використовувати ці API для інтеграції IT-систем або додатків компанії, створення нових систем або програм,

призначених для клієнтів, використовуючи існуючі системи. Навіть якщо програми є загальнодоступними, сам інтерфейс залишається доступним лише для тих, хто працює безпосередньо з видавцем API. Така стратегія дозволяє компанії повністю контролювати використання API. [4]

У будь якому разі API, незалежно від їх типів, спрощують створення програмного забезпечення, і в особливій мірі цьому посприяли віддалені API, а зокрема веб API, що зробили революцію в тому, як саме створюється програмне забезпечення.

Інтерфейс, який використовується у даному випадку браузерним та серверним програмним забезпеченням для спілкування між користувачем і веб ресурсом, з якого користувач бере інформацію, називається веб API. Це по суті обличчя веб сервісу, що безпосередньо приймає та відповідає на запити клієнтів (Рис. 1.2). [5].

Для того, щоб більш точно зрозуміти термін веб інтерфейсу прикладного програмування потрібно розглянути яким саме чином здійснюється обмін даними у всесвітній веб мережі за допомогою протоколу HTTP.

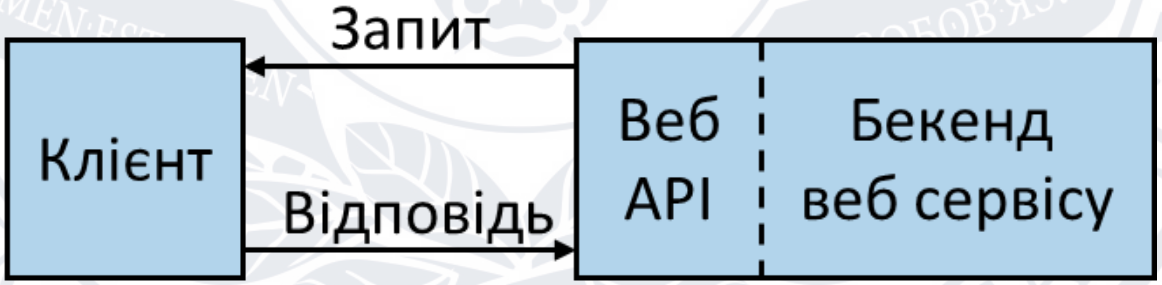


Рис. 1.2. Веб API

1.2. Принцип обміну даними в мережі за допомогою HTTP протоколу

Більшість веб API покладаються на стандартний протокол, щоб переконатися, що машини можуть спілкуватися один з одним через Інтернет.

Цим протоколом є протокол передачі гіпертексту (англ. Hypertext Transfer Protocol) або ж HTTP.

HTTP – це протокол, який використовують веб-браузери та веб-сервери для зв'язку один з одним через Інтернет. [6] Це протокол прикладного рівня, оскільки він розташований на рівні TCP у стеку протоколів (рівень протоколів, що направляє пакет, що в подальшому буде передаватись через мережу до спеціального додатку на комп'ютері, використовуючи номер порту) і використовується певними програмами для спілкування один з одним. У цьому випадку додатками є веб-браузери та веб-сервери.

Найважливіше, що потрібно знати про протокол HTTP, це те, що він був повністю розроблений на основі ідеї передачі повідомлень, а не об'єктів чи результатів запиту до бази даних. Це є однією з причин, чому HTTP так швидко став настільки популярним і настільки поширеним. Поки розробник може написати клієнта або службу, яка може працювати з повідомленнями HTTP, він можете створити HTTP-сервер і розпочати розміщення контенту, незалежно від того, яку мову програмування, об'єктну модель або базу даних він використовує.

Коли клієнт зв'язується із сервером за використовуючи HTTP протокол, відбувається наступне:

1. Веб юзер запрошує TCP-з'єднання із веб-сервером через порт 80.
2. Веб-сервер приймає TCP-з'єднання через порт 80, при цьому попереджаючи користувача.
3. Користувач надсилає HTTP-запит із повідомленням (що містить URL) до TCP-з'єднання, запитуючи потрібні йому об'єкти зі сторінки. Цей запит інкапсулює повідомлення в т. зв. фрейм.
4. Веб-сервер отримує HTTP-запит і відсилає назад HTTP відповідь, який містить об'єкт, що запитувався користувачем (зазвичай HTML (англ. Hypertext Markup Language – мова гіпертекстової розмітки) текст, якщо це веб-сторінка або ж JSON (англ. Javascript Object

Notation) файл, якщо інший еб сервіс). Саме на цьому етапі активну участь бере веб API.

Кожен HTTP запит складається з таких основних частин (Рис. 1.2):

- Метод HTTP – визначає операцію, яку хоче виконати клієнт. Зазвичай клієнт хоче отримати ресурс (за допомогою GET) або опублікувати значення HTML-форми (за допомогою POST), хоча в інших випадках може знадобитися більше операцій.

Багато людей використовують чотири основні методи HTTP (POST, GET, PUT, DELETE) у спосіб схожий для подібних операцій в об’єктно-орієнтованих реляційних базах даних. У цьому підході POST призначений для створення нового запису, GET — для читання існуючого запису, PUT — для оновлення існуючого запису, а DELETE — для видалення запису. Ці чотири методи іноді називають методами CRUD (англ. Create, read, update, delete – створення, читання, оновлення та видалення). [7]

- Шлях отримання ресурсу – URL-адреса ресурсу, вилучена з елементів, які очевидні з контексту, наприклад з протоколу (http://), домену (developer.mozilla.org) або порту TCP (порт 80 на рис.)
- Версія протоколу HTTP
- Додаткові заголовки, які передають додаткову інформацію для серверів
- Тіло для деяких методів, наприклад POST

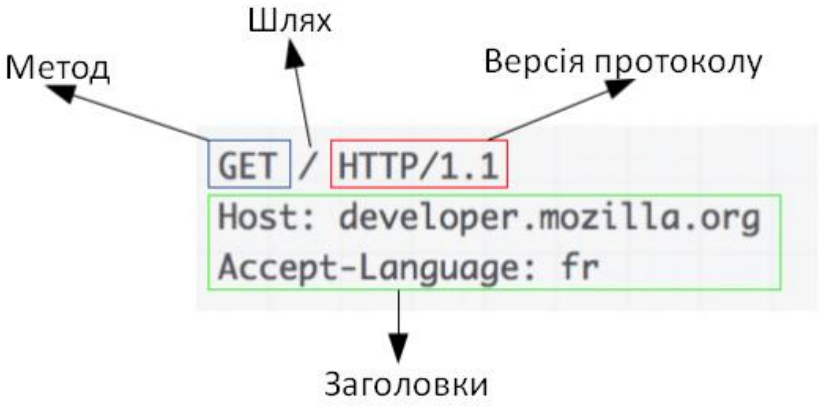


Рис. 1.2. Основні частини, з яких складається HTTP запит

Відповідно, HTTP відповідь складається з наступних елементів (Рис. 1.3):

- Версія протоколу HTTP
- Код статусу, який вказує, чи був запит успішним і у випадку, якщо ні, то чому
- Повідомлення про статус – короткий опис коду статусу
- HTTP-заголовки, як і для запитів

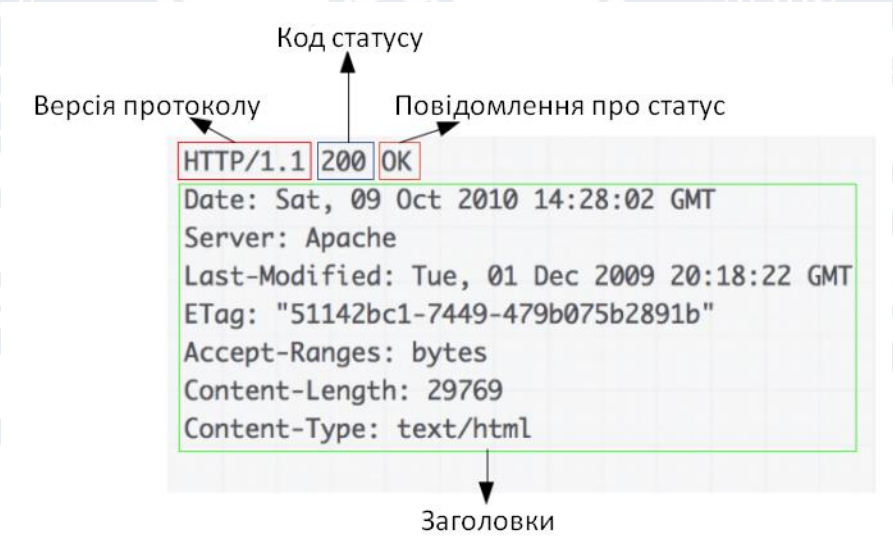


Рис. 1.3. Основні елементи HTTP відповіді

1.3. REST API

Термін REST (англ. Representational State Transfer – передача репрезентативного стану) вперше з’явився у 2000 році і був описаний вченим Роєм Томасом Філдінгом як стиль веб-архітектури. [8]

Незважаючи на те, що REST — це лише стиль, він значно вплинув на те, як саме програмісти розробляють та реалізують свої API як для постачальників послуг, так і для споживачів.

Веб-сервіси — це веб-сервери, які підтримують потреби сайту чи будь-якої іншої веб програми. Архітектурний стиль REST зазвичай застосовується до проектування API для сучасних веб-сервісів. Веб API, що відповідає архітектурному стилю REST, є REST API. [5]

Цей стиль дотримується шести наступних архітектурних обмежень:

- Розподіл клієнта/сервера

Обмеження розподілку клієнту/серверу засноване на принципі чіткого розподілення проблем, коли такі компоненти, як мобільний додаток та API серверу слабо пов’язані один з одним, не знають деталей реалізації один одного та лише комунікують через один спільний інтерфейс. [2]

Завдяки цьому досягаються наступне:

- Відокремлюються проблеми, пов’язані, з одної сторони, з користувацьким інтерфейсом, та, на приклад, із зберіганням даних з іншої.
- Покращується масштабованість серверного обладнання та переносимість на інші платформи клієнтського інтерфейсу
- Дозволяє розвиватися серверній та клієнтській частинам незалежно один від одного

- Відсутність стану

Принцип передбачає використання протоколу «без збереження стану», завдяки якому в період між запитами клієнта на сервері не зберігається інформація про стан клієнта. Кожен клієнтський запит повинен містити усю інформацію, необхідну для того, щоб сервер міг зрозуміти запит, оскільки він не може використати інформацію з попередніх запитів, так як не запам’ятовує їх. Таким чином, уся інформація про стан сесії цілком і повністю зберігається на стороні клієнта. [9]

Це обмеження дозволяє покращити такі властивості системи, як видимість, надійність та масштабованість. Видимість покращується, оскільки системі не потрібно розглядати інформацію за межами запиту, для того щоб встановити повну природу запиту. Надійність покращується, оскільки задача відновлення після системних збоїв значно спрощується. Щодо масштабованості, то обмеження дозволяє не зберігати серверу інформацію про стан сесії, що додатково вивільняє ресурси та збільшує продуктивність і зникає необхідність керувати використанням ресурсів поза межами клієнтських запитів.

Відповідно, очевидним недоліком такого архітектурного рішення є зниження продуктивності мережі за рахунок збільшення повторюваних даних, що надсилаються серією запитів, так як ці дані не залишаються на сервері.

- Кешування

В якості вирішення проблеми недоліку обмеження відсутності інформації про стан сесії на серверній компоненті Роєм Філдінгом було запропоновано використати процес кешування. Він дозволяє збільшити ефективність мережі завдяки уникненню повторних обмінів даними між клієнтом та сервером для отримання того ж самого ресурсу.

Обмеження кешування потребує явного або неявного маркування даних у відповіді на клієнтський запит як кешованих та некешованих. Якщо відповідь кешована, тоді кешові клієнта надається можливість повторно використати дані з відповіді для подальших еквівалентних запитів. [9]

Перевагою обмеження додавання кешу є усунення частини взаємодій, що підвищує ефективність, масштабованість та продуктивність, як з боку сервера, так і клієнта за рахунок зменшення середнього часу затримки серії взаємодій.

Недоліком є зменшення надійності системи через можливість надсилання застарілих даних у кешованому запиті.

- Єдиний/однорідний інтерфейс

Єдиний, або ж однорідний інтерфейс є центральною особливістю, яка відрізняє архітектурний стиль REST від інших мережевих стилів. Він дозволяє спростити загальну архітектуру системи та покращити видимість взаємодій між компонентами.

REST визначається чотирма обмеженнями інтерфейсу: ідентифікація ресурсів; маніпуляція ресурсами через їх представлення; повідомлення з самоописом і гіпермедіа як рушій стану програми.

- Ідентифікація ресурсів

Кожен ресурс повинен мати власний унікальний ідентифікатор. Прикладом є HTTP URI (англ. Uniform Resource Identifier/Locator – єдиний ідентифікатор ресурсу). REST API розроблено навколо ресурсів, які є будь-якими об’єктами, даними або послугами, до яких може отримати доступ клієнт. У відповідь на запит клієнта надається ресурс у певному представленні. Це може бути HTML, XML (англ. eXtensible Markup Language – розширена мова розмітки), JSON, TXT та ін.

- Маніпуляція ресурсами через представлення

Маніпуляція ресурсами через представлення дозволяє клієнтам і серверам контролювати представлення кожного запиту та відповіді за допомогою форматів повідомлень.

- Повідомлення із самоописом

Кожен запит і відповідь повинні містити усю необхідну інформацію для його обробки. Прикладом такої інформації в HTTP повідомленні є код статусу HTTP, хост і т. д.

- Гіпермедіа як рушій стану застосунку

Зміна даних, що зберігаються на серверній частині, повинна здійснюватися за допомогою гіперпосилань або форм, які надають та отримують достатньо інформації для взаємодії між користувацьким і серверним інтерфейсом.

- Багатошарова система

Стиль багатошарової системи дозволяє архітектурі складатися з ієрархічних рівнів, що обмежують поведінку компонентів таким чином, що кожен компонент не може «бачити» за межі безпосереднього шару, з яким вони взаємодіють. [9]

Обмежуючи знання системи одним шаром, складність її компонентів значно полегшується. Також, шари дають можливість інкапсулювати застарілі служби і захистити служби від взаємодії із застарілими клієнтами. Окрім того, використання багатошарової системи дозволяє впроваджувати елементи політики безпеки, наприклад, брандмауер. Таким чином, покращується масштабованість, надійність та гнучкість системи.

- Запитування коду

Останнім, шостим обмеженням архітектурного стилю REST є запитування коду. Клієнтські програми мають бути розширюваними за допомогою використання додатково завантажуваного коду замість того, щоб створювати нові клієнтські програми для кожної нової функції. Це обмеження є необов’язковим і якщо його впровадження не надає переваг веб додатку, то його краще уникати, оскільки завантаження стороннього коду може негативно вплинути на систему з точки зору порушення її безпеки.

Підсумовуючи усю інформацію, REST API виконує ряд CRUD операцій з веб ресурсами доступ до яких здійснюється через унікальний ідентифікатор за допомогою HTTP протоколу.

1.4. Особливості забезпечення безпеки API сервісів

Питання безпеки наразі є основною проблемою для організацій, які займаються розробкою, тестуванням та експлуатацією API сервісів. Безпека конфіденційних даних, будь то корпоративна або персональна інформація, є важливим фактором. API не є винятком, оскільки вони керують тими частинами клієнтських та серверних систем, що здійснюють операції над ресурсами, що потребують захисту від загроз та порушень безпеки.

Безпека API знаходиться на перетині декількох дисциплін забезпечення безпеки (Рис. 1.4). [10] Найважливішими із них є:

1. Інформаційна безпека

Інформаційна безпека пов’язана із захистом інформації напротязі її повного життєвого циклу – створення, збереження, передачі, резервного копіювання і її остаточне знищення.

Інформаційна безпека дозволяє:

- Визначити цілі впровадження безпеки щодо API сервісів і виявляти загрози
- Здійснювати захист API використовуючи техніку контролю доступу
- Здійснювати захист інформації використовуючи криптографічні алгоритми

2. Безпека мережі

Безпека мережі займається захистом як даних, що передаються через неї так і запобіганням неавторизованого доступу до самої мережі.

Безпека мережі дозволяє впровадити наступні заходи:

- Базову інфраструктуру захисту веб API, що включає брандмауер, балансувальники навантаження (англ. load-balancers), зворотні проксі (англ. reversed proxies).
- Безпечні протоколи обміну інформацією, а саме HTTPS (англ. HyperText Transfer Protocol Secure) для захисту даних, що передаються з допомогою API.

3. Безпека додатків

Безпека додатків гарантує, що програмні системи розроблені та створені так, щоб протистояти атакам і неправомірному використанню.

Безпека додатків впроваджує наступні заходи:

- Методи безпечного кодування.
- Захист від відомих від відомих вразливостей ПЗ.
- Безпечний засіб збереження і керування системою та обліковими даними користувача, що необхідні для отримання доступу до API.

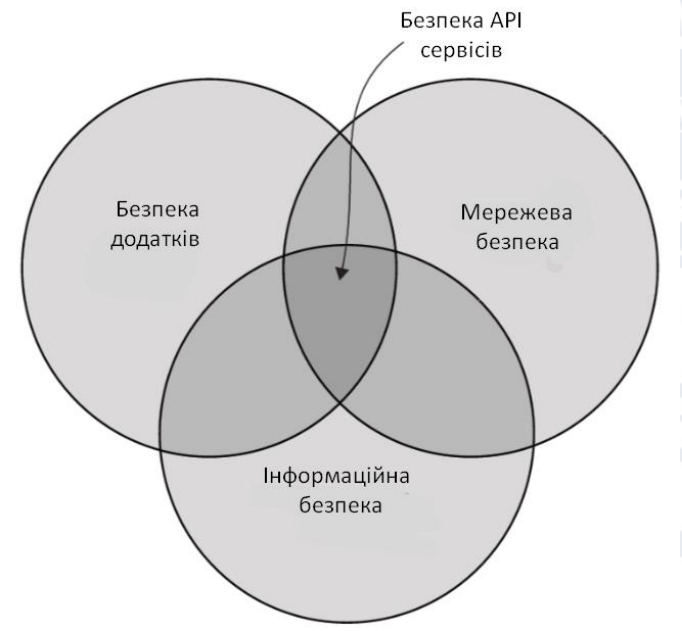


Рис. 1.4. Безпека API сервісів як перетин трьох дисциплін забезпечення безпеки

1.4.1. Аналіз основних загроз безпеки API сервісів

Згідно проекту OWASP (англ. Open Web Application Security Project – відкритий проект забезпечення безпеки веб-додатків) найбільш серйозними загрозами безпеки API є: [11]

1. Зламана авторизація на рівні об’єкту

Авторизація на рівні об’єкту – механізм контролю доступу, що зазвичай впроваджений на рівні коду для підтвердження, що користувач може отримати доступ лише до тих об’єктів, які йому дозволені.

Зловмисники можуть використати кінцеві точки API сервісу, що є вразливими даного типу атаки. Кінцеві точки API сервісу – це звернення до маршруту (URL) окремим HTTP методом. Кінцеві точки приймають від користувача параметри та повертають дані у відповідь на запит клієнта. Використовуючи вразливості таких кінцевих точок, можна маніпулювати ID об’єкта, що надсилається у запиті. Відповідно, це може призвести до неавторизованого доступу до конфіденційних даних. Атаки такого типу є досить розповсюдженими у API сервісах, оскільки сервер не відслідковує стан клієнта (обмеження «відсутність стану») і покладається на параметри клієнтського запиту (в цьому випадку – ID) для того щоб зрозуміти, до які об’єкти ресурсів надавати у відповіді.

Запобіжними методами є:

- Впровадження належних механізмів авторизації згідно користувацьких політик і ієрархії.
- Використання таких механізмів до зареєстрованих користувачів з метою перевірки прав доступу параметризованого запиту користувача до відповідного об’єкту.
- Використання випадкових та складних ідентифікаторів для об’єктів
- Регулярне тестування механізму авторизації

2. Зламана автентифікація користувача

Реалізація автентифікації у API сервісах достатньо складна задача. Механізми автентифікації публічних API – це легка мішень для зловмисника, оскільки у таких API відкритий вихідний код.

API вважається вразливим, якщо:

- Дозволяє заповнення облікових даних (англ. credentials stuffing).

Заповнення облікових даних – це автоматичне введення вкрадених або розкритих облікових даних у форми входу на веб сайти, щоб отримати доступ до облікових записів користувачів. [12]

- Дозволяє брут-форс атаки, тобто у механізму автентифікації API сервісу не впроваджені капча або ж механізм блокування облікового запису.
- Не перевіряє токени автентифікації.
- Надсилає конфіденційні дані (такі як токени автентифікації або паролі) вказуючи їх в URL.
- Дозволяє використання слабких паролів
- Не перевіряє підписи JWT токенів (англ. JSON Web Token – веб токен у форматі JSON) або термін їх дії. Такі токени як правило передають дані автентифікації і повинні підписуватись секретним ключем перед їх передачею користувачеві.
- Використовує незашифровані або слабко хешовані паролі
- Використовує слабкі алгоритми або ключі шифрування

Запобіжними методами є:

- Використання перевірки на слабкі паролі.
- Використання багатофакторної автентифікації.
- Використання існуючих стандартів в автентифікації, генерації токенів та зберіганні паролів.

- Кінцеві точки полів відновлення паролю також повинні захищатись від брут-форс атак, атак обмеження швидкості (DoS атаки) та ін.
- Використання механізмів протидії атак типу брут-форс на кінцеві точки автентифікації, щоб пом’якшити атаки заповнення облікових даних, словникових атак, брут-форс та ін.
- Пам’ятати, що API ключі використовуються для автентифікації клієнтського проекту/додатку, а API токени – для автентифікації користувача, а не навпаки.
- Ніколи не надсилати облікові дані через незахищене з’єднання, а також не розкривати ідентифікатори сеансу у веб-URL. [13]

3. Надмірне розкриття даних

API сервіси побудовані таким чином, що вони повинні повертати конфіденційні дані, що зазвичай фільтруються на стороні клієнта перед тим, як подаватись користувачеві. Зловмисник може використати сніфер щоб зчитати трафік та отримати конфіденційні дані.

Проблема постає в тому, що автоматичні інструменти зазвичай не здатні виявити цей тип вразливості, оскільки таким інструментам важко відрізнити легітимні дані, що повертаються API сервісами, і конфіденційні дані, які не слід повертати без глибокого розуміння програми.

Запобіжні методи проти цієї атаки наступні:

- Не покладатись на клієнтську сторону що фільтрує конфіденційні дані.
- Перевіряти відповіді, що надсилаються через API сервіси з метою перевірки надсилання лише легітимних даних.
- Впровадити механізм валідації даних, що повертаються усіма методами API, включаючи помилки, як додатковий рівень безпеки.

4. Нестача ресурсів та обмеження швидкості (англ. rate-limiting)

Запити та відповіді API сервісів споживають системні ресурси серверних компонент, на яких розміщенні дані. Кількість ресурсів, необхідних для задоволення запиту, значною мірою залежить від даних, що відправлені у запиті користувача та бізнес-логіки кінцевої точки. Також важливо пам’ятати, що безліч запитів клієнтських API постійно ділять між собою серверні ресурси.

API вважається вразливим, якщо хоча б один із елементів системних ресурсів налаштований неправильно. Такими елементами є:

- Час очікування виконання запиту
- Максимальна доступна пам’ять (як постійна, так і тимчасова)
- Кількість процесів
- Кількість одночасних запитів на клієнта/ресурс
- Розмір даних, що надсилаються у клієнтському запиті та даних, що запитуються, у відповіді

Зловмисник може надіслати безліч простих запитів, що не потребують автентифікації зі сторони клієнта. Безліч одночасних запитів можна виконувати з одного локального комп’ютера або за допомогою хмарних обчислювальних ресурсів.

Експлуатація даної вразливості може призвести до DoS і, як наслідок, API може перестати реагувати або ж і зовсім стане недоступним.

Відповідно, щоб уникнути атаки обмеження швидкості рекомендується:

- Використання Docker для налаштування обмеження серверних ресурсів.
- Введення обмеження на те, як часто клієнт може викликати API сервіс за проміжок часу.
- Введення перевірки параметрів запиту зі сторони серверу, в особливості параметрів, що відповідають за кількість записів, що повернуться у відповіді.

- Визначити максимальний розмір даних, що надсилаються у запиті та повертаються у відповіді.

5. Зламана авторизація на функціональному рівні

Зловмисник може експлуатувати вразливість надсилаючи дозволені API виклики до кінцевих точок API, доступу до яких він не повинен мати. Ці кінцеві точки можуть бути доступні для анонімних або звичайних, непривілейованих користувачів. Такі недоліки зловмисникам нескладно виявити, оскільки API є структурованими, а спосіб доступу до певних функцій більш передбачуваний (наприклад, заміна методу HTTP з GET на PUT або рядка «users» в URL на «admins»).

Такі недоліки дозволяють зловмисникам отримати доступ до неавторизованих функцій. Адміністративні функції є ключовими мішенями цього типу атаки.

Запобіжними методами є:

- Заборона відповідними механізмами безпеки будь-якого доступу, вимагаючи явного надання певних ролей для доступу до кожної функції.
- Перевірка кінцевих точок на наявність недоліків авторизації на рівні функцій.

6. Масове призначення

Об’єкти в додатках зазвичай мають ряд властивостей, що несуть як відкриту, так і конфіденційну інформацію про об’єкт. Відповідно, деякі властивості можуть бути змінені користувачем (наприклад, ім’я або адреса користувача), а деякі лише адміністраторами або ж автоматично самим додатком (права доступу користувача).

Кінцеві точки API сервісів є вразливими, якщо вони автоматично перетворюють параметри, що можуть змінюватися клієнтом у внутрішні властивості об’єкта, не враховуючи конфіденційність і рівень відкритості цих

властивостей. Це може дозволити зловмиснику змінювати властивості об’єкта, до яких він не повинен мати доступу.

Приклади закритих властивостей:

- Властивості на основі дозволу – права доступу, що змінюються лише адміністраторами.
- Властивості, що залежать від процесів, керованих системою – наприклад, зміна властивості, що відповідає за суму на рахунку клієнта після проведення оплати.
- Внутрішні властивості – наприклад, дата публікації статті, яка виставляється системою.

Експлуатація вразливості може призвести до ескалації привілеїв, підробки даних, обходу механізмів безпеки тощо.

Превентивними заходами є:

- Уникання використання функцій, які автоматично прив’язують ввід клієнта до змінних коду або інших внутрішніх об’єктів.
- Створення білого та чорного списків властивостей об’єкту.

7. Неправильна конфігурація налаштувань безпеки

API може бути вразливим, якщо:

- Використовуються застарілі системи безпеки.
- Увімкнено непотрібні функції (наприклад, деякі із методів HTTP).
- Відсутній протокол безпеки транспортного рівня, або ж TSL (англ. Transport Security Layer).
- Директиви безпеки, напрямлені на підвищення поінформованості клієнтів щодо основних правил безпеки, не надсилаються користувачам.
- Повідомлення про помилки містять трасування стека або розкривається інша конфіденційна інформація.

Неправильна конфігурація налаштувань безпеки може розкрити не тільки конфіденційні дані користувача, але й деталі самої системи, що може призвести до повної компрометації сервера.

Запобіжними заходами вважаються такі дії на протязі життєвого циклу API сервісу:

- Регулярний перегляд та оновлення конфігурацій у всьому стеку API. Перевірка має включати: системні файли, компоненти API та хмарні сервіси.
- Безпечний канал зв’язку для всіх клієнт-серверних взаємодій API сервісів.
- Автоматизований процес для постійної оцінки ефективності конфігурації та налаштувань у всіх середовищах.

8. Ін’єкції

Зловмисники згодовують API шкідливі дані через будь-які доступні вектори ін’єкції (наприклад, пряме введення, параметризовані запити, інтегровані служби тощо), що у незахищеній системі будуть надіслані інтерпретатору.

Ін’єкція може призвести до розкриття інформації та втрати даних. Це також може призвести до DoS або повного контролю хоста зловмисниками.

API вважається вразливим до атак через ін'єкції у випадку, якщо:

- Дані, що надсилає клієнт, не перевіряються та не фільтруються.
- Дані, що надаються клієнтом, безпосередньо використовуються або об’єднуються із запитами SQL/NoSQL, командами ОС, синтаксичними аналізаторами XML та ін. елементами системи.
- Дані, що надходять із зовнішніх систем (наприклад, інтегрованих систем), не перевіряються та не фільтруються.

З метою запобігання атакам типу ін’єкцій використовуються наступні превентивні заходи:

- Зберігання даних окремо від команд та запитів
- Здійснення перевірки та фільтрації даних, що направляються користувачем або інтегрованою системою до API серверної частини
- Використання параметризованого інтерфейсу
- Обмеження кількості запитів що повертаються задля запобігання масовому розголошенню розкритої інформації у випадку ін’єкції
- Визначення типів даних і суворі шаблони для всіх строкових параметрів.

9. Неналежне управління активами

Старі версії API можуть містити невиправлені вразливості, що можуть скомпрометувати систему навіть без необхідності обходу найсучасніших механізмів безпеки, які можуть застосовуватись для захисту лише останніх версій API сервісів.

Зловмисники можуть отримати доступ до конфіденційних даних або навіть отримати контроль над усім сервером за допомогою старих невиправлених версій API, що, наприклад, підключених до однієї бази даних.

Для запобігання цьому необхідно:

- Визначити усі хости API сервісу, включаючи його більш старі версії та задокументувати усі важливі особливості
- Використовувати зовнішні заходи захисту, такі як брандмауери безпеки API для всіх відкритих версій API, а не лише для поточної робочої версії.
- Виконувати аналіз ризиків кожен раз, коли API отримує оновлення заходів безпеки, щоб зрозуміти, чи сумісні вони з більш старими версіями API.

10. Недостатнє ведення журналу та моніторинг

Зловмисники можуть експлуатувати відсутність системи моніторингу та логування, щоб непомітно зловживати API сервісом. Без цих механізмів

захисту майже неможливо відстежити підозрілу активність у системі та вчасно на неї відреагувати. Не маючи видимості щодо поточної шкідливої діяльності, у зловмисників є достатньо часу, щоб повністю скомпрометувати систему.

API сервіс є вразливим, якщо:

- Не створюються журнали логування, або ж рівень їх ведення встановлено неправильно, або повідомлення журналу містять недостатню інформацію.
- Цілісність журналу може бути порушена.
- Журнали не переглядаються регулярно адміністраторами системи.

Заходами запобігання експлуатації вразливості є:

- Логування усіх невдалих спроб автентифікації, повідомлень заборони дозволу або помилок недійсних вхідних параметрів.
- Журнали мають бути записані у форматі, придатному для використання автоматизованим ПЗ (наприклад системи керування подіями) для керування журналами, і мають містити достатньо деталей, щоб такі системи могли ідентифікувати зловмисника.
- До журналів потрібно відноситись як до конфіденціальної інформації, тому їх цілісність необхідно захищати на тому ж рівні, як і всю іншу чутливу інформацію.

1.4.2. Аналіз механізмів безпеки API сервісів

Задля протидії загрозам, описаним у пункті 1.5.1 API сервіси реалізовують ряд захисних механізмів.

Найбільш застосовуваними механізмами захисту API є:

- Шифрування
- Ідентифікація та автентифікація
- Контроль доступу

- Аудит та логування
- Обмеження швидкості

На рисунку 1.5 [10] зображено п’ять зазначених процесів у вигляді ряду фільтрів, через які проходить запит перед тим як його обробить основна логіка API сервісу.

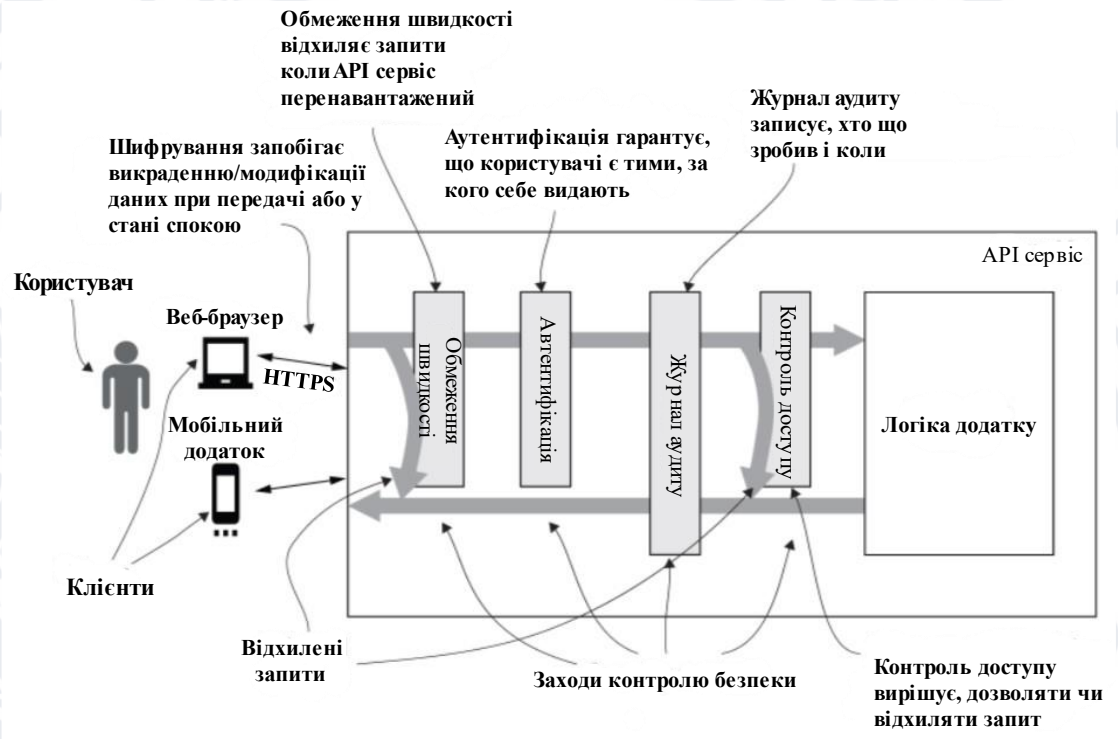


Рис. 1.5. Багатошарова система забезпечення безпеки API сервісу та даних, що ним керуються [10]

Спочатку запити та відповіді шифруються за допомогою протоколу HTTPS і/або іншими криптографічними алгоритмами. Обмеження швидкості застосовується для запобігання атак типу DoS. Далі, клієнти виконують процес ідентифікації та автентифікації, а спроба доступу, незалежно від того, успішна вона, чи ні записується у журналі аудиту. Нарешті, проводиться перевірка прав доступу, щоб вирішити, чи має цей користувач можливість виконати той чи інший запит. Результат запиту також має бути зафіксований у журналі аудиту.

1.4.2.1. Шифрування даних та протокол HTTPS

Шифрування даних необхідне щоб захистити дані, поки вони знаходяться за межами API, тобто коли запити та відповіді передаються через мережу Інтернет. Також важливо зауважити, що необхідність шифрування виникає для забезпечення цілісності та конфіденційності даних, що зберігаються у постійні пам’яті на серверній компоненті. Для захисту інформації, що міститься в запитах і відповідях при передачі найчастіше використовується протокол безпечної передачі гіпертексту HTTPS (англ. HyperText Transfer Protocol Secure).

HTTPS розроблено, щоб зробити протокол HTTP безпечним, дозволяючи йому працювати поверх протоколу SSL/TLS. [14] URL HTTPS починається з «https://» і використовує порт 433 за замовчуванням.

Існують два важливі аспекти, пов’язані з використанням HTTPS: по-перше, він гарантує, що користувач спілкується саме з тим веб-сайтом, до якого він має намір отримати доступ, і що сайт, до якого користувач підключений, не маскується під інший сайт, якому він довіряє. По-друге, це не дозволяє будь-кому посередині читати або змінювати повідомлення, якими обмінюються клієнт та сервер. HTTPS реалізовує це, шифруючи все повідомлення, включаючи URI запиту, заголовки та файли cookie. З цієї причини деякі речі, як-от кешування, не працюватимуть із HTTPS.

1.4.2.2. Ідентифікація та автентифікація

Ідентифікація та автентифікація в API сервісах може здійснюватися трьома різними способами: за допомогою HTTP, API ключів та/або токенів та за допомогою відкритого протоколу авторизації OAuth.

- Автентифікація за допомогою HTTP

Одним з найпростіших рішень є базова аутентифікація за допомогою протоколу HTTP. У цьому підході користувач HTTP просто надає свій ідентифікатор та пароль, щоб підтвердити автентифікацію. Цей підхід не вимагає файлів cookie, ідентифікаторів сеансів, сторінок входу та інших подібних спеціальних рішень, а оскільки він використовує сам заголовок HTTP.

Проблема такої автентифікації в тому, що якщо процес суворо не виконується протягом усього циклу передачі даних за допомогою SSL, аутентифікація передається у відкритому виді по незахищеним лініям зв'язку. Це можуть використати, наприклад, для атаки людини посередині (англ. man in the middle attack), коли користувач може просто захопити облікові дані та автентифікуватися за допомогою HTTP-заголовка, приєднаного до шкідливого пакету.

Окрім того, навіть якщо SSL і застосовується при передачі, це призводить до уповільнення часу відповіді на запит. І навіть якщо це ігнорувати, HTTP у своїй базовій формі жодним чином не шифрується, а при використанні протоколу HTTPS передати облікові дані системі неможливо.

- API ключі та токени

Токени та ключі API були створені як рішення проблем базової автентифікації за допомогою HTTP та інших схожих систем. У цьому підході кожному користувачеві, що вперше авторизується у системі, призначається унікальне згенероване значення, що підтверджує його особистість. Таке значення називається API токеном. Коли користувач намагається повторно увійти в систему, його токен використовується для підтвердження того, що це той самий користувач, що й раніше.

API ключ – це також унікальний ідентифікатор, але він використовується для ідентифікації стороннього проекту, що має спільний інтерфейс взаємодії із API сервісом.

З одного боку, це дуже швидко. Можливість підтвердити особистість один раз є дуже гнучкою, і саме тому вона вже багато років використовується як підхід за замовчуванням для багатьох постачальників послуг API. Крім того, налаштування такої систему це дуже швидкий та простий процес, а керувати згенерованими ключами та токенами ще простіше.

Проблема, однак, у тому, що ключі API часто використовуються для того, чим вони не є. Ключі та токени API це не методи не авторизації, а аутентифікації. Оскільки будь-хто, хто робить запит на послугу, передає свій токен або ключ, то теоретично його може бути отримано так само легко, як і будь-які інші дані, що передаються по мережі, і якщо якась точка у всій мережі небезпечна, вся мережа підпадає під ризик.

- OAuth

Можна сказати, що OAuth – це комбінація перших двох методів, оскільки він також працює із токенами, але при цьому також здійснює роботу з обліковими даними користувача. Він працює за принципом делегування аутентифікації користувача сервісу, на якому знаходиться обліковий запис користувача, що дозволяє сторонньому додатку отримувати доступ до облікового запису користувача.

Алгоритм роботи протоколу наступний:

1. Клієнт запитує користувача авторизацію на доступ до сервера ресурсів.
2. Якщо користувач авторизує запит, клієнт отримує дозвіл на авторизацію.
3. Програма запитує токен авторизації у API сервісу шляхом надання інформації про себе і дозвіл на авторизацію від користувача.
4. Якщо автентифікація клієнта підтверджена і дозвіл на авторизацію дійсний, API сервіс створює токен доступу для клієнта.
5. Клієнт запитує конкретний ресурс, надаючи при цьому токен доступу для автентифікації.

6. Якщо токен дійсний, сервер ресурсів надає запитаний ресурс клієнту.

Цей підхід до авторизації є значно безпечнішим ніж просте використання токенів чи базової HTTP авторизації і єдиним його недоліком серед двох попередніх способів є більш складна реалізація і використання ресурсів.

1.4.2.3. Контроль доступу і авторизація

Існує два основних підходи до контролю доступу, які використовуються для API: [10]

• Контроль доступу на основі ідентифікації

Контроль доступу на основі ідентифікації спочатку ідентифікує користувача, а потім визначає, що він може робити, виходячи з того, хто він є. Користувач може спробувати отримати доступ до будь-якого ресурсу, але йому може бути відмовлено в доступі відповідно до правил контролю доступу.

Цей підхід вводить такі заходи захисту, як:

- Організація користувачів у групи
- Спрощення дозволів за допомогою контролю доступу на основі ролей
- Реалізація складних політик за допомогою контролю доступу на основі атрибутів(набору характеристик об’єкту)

• Контроль доступу на основі можливостей

Контроль доступу на основі можливостей використовує спеціальні маркери або ключі, відомі як можливості доступу до API. Сама можливість доступу визначає, які операції може виконувати носій, а не хто є користувачем. Можливість доступу одночасно називає ресурс і описує дозволи до нього, тому користувач не може отримати доступ до жодного ресурсу, доступ до якого можливостями не описаний.

1.4.2.4. Обмеження швидкості

Існує безліч способів обмеження швидкості API сервісів. Найбільш популярними серед них є:

1. Черги запитів

Існує безліч бібліотек , що спрощують реалізацію черг запитів, і кожна мова програмування або середовище розробки має свої команди. Такі бібліотеки зазвичай досить легко та швидко інтегрувати із API системою.

Наприклад, за допомогою одної бібліотеки можна встановити обмеження швидкості на рівні двох запитів на секунду і далі викликати помилку, інша ж поміщає їх у спеціальну чергу запитів.

2. Дроселювання (англ. throttling)

Дроселювання – спосіб контролю рівня трафіку, що дозволяє контролювати, як використовується API шляхом налаштування стану, що дозволяє API оцінювати кожен клієнтський запит. У випадку, якщо запит не пройшов таку перевірку, користувач може бути відключений, або може бути просто зменшена його пропускна здатність.

Цей спосіб дозволяє регулювати трафік як на програмному рівні, так і на рівні API і користувацькому рівні.

3. Алгоритми обмеження швидкості

Існує безліч алгоритмів обмеження швидкості. Найрозповсюдженішими із них є алгоритм Leaky Bucket, Fixed Window та Sliding Window.

Алгоритм Leaky Bucket – переводить обробку запитів у формат «перший надійшов – перший вийшов» (англ. FIFO – first in-first out), що дозволяє обробляти елемент у черзі зі звичайною швидкістю.

Алгоритм Fixed Window – використовує звичайний лічильник, за допомогою якого операцією інкрементації підраховує кількість запитів за

певний встановлений проміжок часу. Якщо, наприклад, фіксоване максимальне значення кількості запитів, які може надіслати користувач за годину 3600 і користувач перевищує це значення, трафік, що від нього надходить сповільнюється або ж і зовсім зупиняється.

Алгоритм Sliding Log – записує запити від кожного користувача у спеціальні журнали з часовими помітками. Якщо ці часові помітки перевищують встановлений системою ліміт, запити поміщаються в чергу або ж користувач отримує помилку.

Алгоритм ковзного журналу включає відстеження кожного запиту через журнал із позначкою часу. Журнали з часовими відмітками, що перевищують ліміт швидкості, відкидаються.

РОЗДІЛ 2. ОГЛЯД ІСНУЮЧИХ ІНСТРУМЕНТІВ ТЕСТУВАННЯ МЕХАНІЗМІВ ЗАБЕЗПЕЧЕННЯ БЕЗПЕКИ API СЕРВІСІВ

Для забезпечення безпеки REST API сервісів використовуються інструменти автоматизованого тестування, що дозволяють виявити існуючі вразливості у механізмах, за допомогою яких реалізовується безпека веб API, описаних у Розділі 1:

1. Ідентифікація та автентифікація
2. Контроль доступу
3. Аудит та логування
4. Обмеження швидкості

2.1. Аналіз інструментів тестування безпеки механізмів автентифікації у REST API

2.1.1. Тестування базової автентифікації HTTP за допомогою Burp Intruder

Двома головними вразливостями автентифікації за допомогою HTTP є те, що цей спосіб ніколи не блокує багаторазові повторні спроби і передає облікові дані при передачі запиту у вигляді простого тексту. Це розкриває вразливість до атак типу заповнення облікових даних, брут-форс атак та ін.

Для того, щоб здійснити тестування API сервісу на наявність цих двох вразливостей, можна використати інструмент Burp Intruder пакету інструментів тестування на проникнення Burp Suite.

Режим роботи Burp Suite Intruder зазвичай здійснюється через HTTP-запит і дає можливість будь-яким чином змінити цей запит. Також, цей інструмент можна використовувати для аналізу відповідей програми на

запити, що дає можливість використовувати його для перевірки вразливостей не лише засобу базової автентифікації за допомогою HTTP, але і інших двох способів також. [15]

Для того, щоб здійснити брут-форс атаку за допомогою Burp Intruder, тестувальнику необхідно у пакеті запиту помітити логін та пароль як параметри ін'єкції. Наступним кроком, необхідно надіслати до цільового об'єкту безліч запитів із потенційними обліковими даними (які можна отримати із безлічі різних баз даних розкритих облікових даних) (Рис. 2.1) та проаналізувати ті пакети, які мають код 200 (успішно виконаний запит).

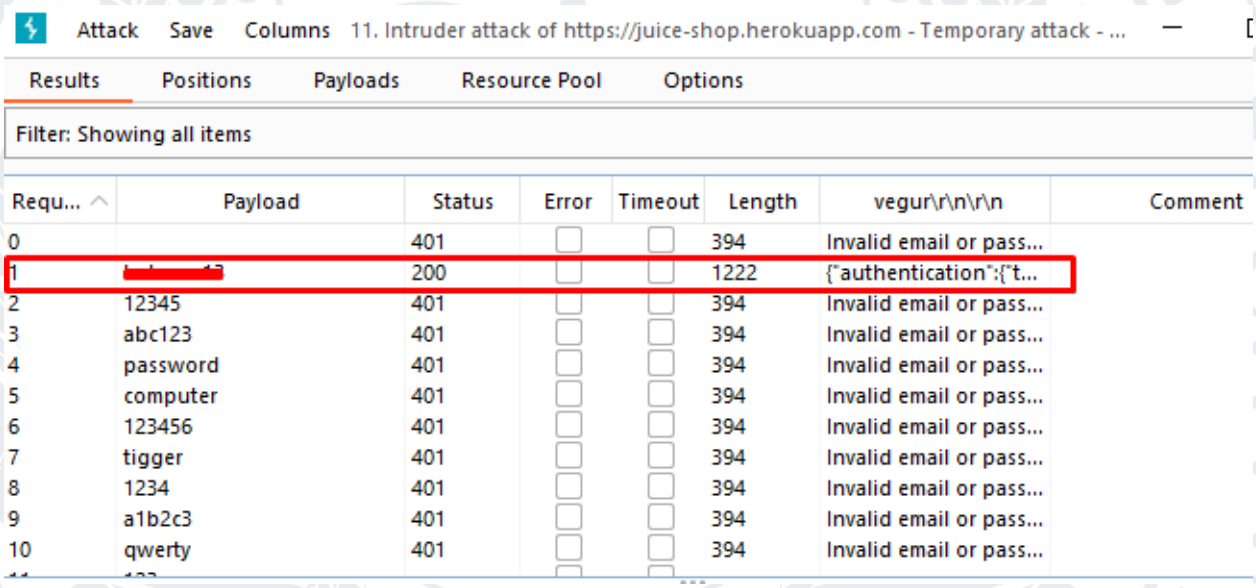


Рис. 2.1. Відповіді на брут-форс запити із вставленими обліковими даними

2.1.2. Тестування безпеки API токенів за допомогою JWT-cracker

Режим автентифікації за допомогою API токенів вимагає від запитів користувача попередньо згенерованого унікального API токену. Найчастіше за все, у REST API сервісах використовуються JWT токени.

JWT токен складається з трьох частин [16]:

- Заголовок, в якому міститься така інформація, як алгоритм шифрування секретної частини токену та тип токену.
- Фактична інформація про об’єкт (англ. payload), що може містити таку інформацію, як власник токену, його права доступу, термін придатності та ін.
- Підпис, необхідний для шифрування токену.

Використовуючи вразливість підпису JWT, можна підробити токен і за допомогою нього здійснити доступ до API сервісу. Для цього необхідно спочатку проаналізувати токен, що видається API сервісом за допомогою послуги з офіційного сайту JWT, що дозволяє розшифрувати відкриту частину токену і дізнатись алгоритм шифрування підпису (Рис. 2.2). З рисунку видно, що алгоритм шифрування токену HS256 (симетричний алгоритм шифрування).

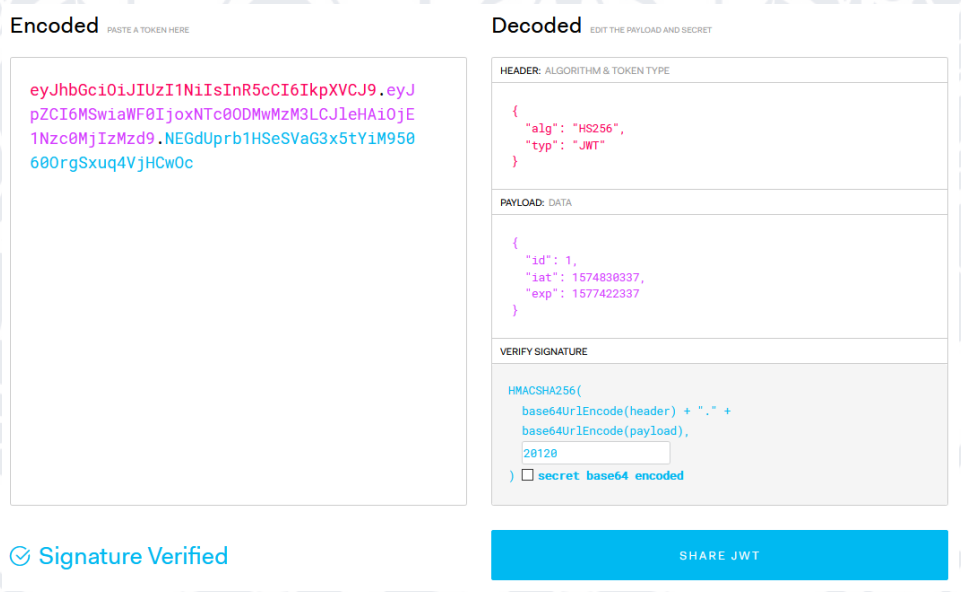


Рис. 2.2. Дешифрування відкритої частини JWT токена

Знаючи алгоритм шифрування і у випадку, якщо він слабкий, можна здійснити спробу розшифровки, у випадку для алгоритму HS256 можна використати брут-форс атаку. Для цього можна використати утиліту для Kali Linux JWT-cracker (Рис. 2.3).

```
root@attackdefense:~# jwt-cracker eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Miwi
aWF0IjoxNTc0ODMwMzM3LCJleHAiOjE1Nzc0MjIzMzd9.CBt2d7VDOoyydyGZgIy3uTXQfvA0EbAH5z8Hxq
3TiYk 1234567890 6
Attempts: 100000
SECRET FOUND: 20120
Time taken (sec): 1.368
Attempts: 102202
root@attackdefense:~#
```

Рис. 2.3. Використання JWT-cracker для розшифровки секретної частини JWT токену

З рис. 2.3 видно, що ключем, що використовувався для підпису токену є 20120. Тепер, знаючи підпис, можна підробити будь-який токен і здійснити авторизацію через API сервіс.

2.1.3. Використання OAuth.Tools для тестування безпеки роботи протоколу OAuth 2.0

OAuth.Tools – це інструмент, що дозволяє проаналізувати рівень безпеки та потенційні вразливості використання протоколу OAuth 2.0 для авторизації у REST API сервісах. За допомогою нього можна дослідити безпеку усіх етапів роботи реалізованого на API сервері протоколу, серед яких:

1. Передача запитів на сервер авторизації API та отримання відповідей із т. зв. кодом авторизації.
2. Обмін кодом авторизації для отримання токену доступу.
3. Передача запиту з токеном доступу та його перевірка для отримання захищених ресурсів з API серверу.
4. Оновлення токену доступу із закінченням терміну його придатності.

2.2. Тестування безпеки контролю доступу

Основною вразливістю контролю доступу у веб API є ескалація привілеїв – від незареєстрованого користувача до зареєстрованого та від звичайного зареєстрованого до адміністратора. Цього можливо досягнути різними способами, найрозповсюдженішими з яких є:

- Модифікація URL
- Ескалація привілеїв – від незареєстрованого користувача до зареєстрованого, від звичайного користувача до адміністратора
- Маніпуляція метаданими (через зміну JWT токена, токена доступу та ін.)
- Доступ до API через відсутність контролю доступу для POST, PUT та DELETE

В усіх вразливостей контролю доступу є одна головна спільна риса – зловмисники використовують ін’єкцію параметрів у запит до API сервісу.

Для такої ін’єкції можна знову ж таки використати інструменти пакету Burp Suite Burp Intruder та Burp Repeater. Burp Repeater дозволяє змінити запит через Burp Intruder та надіслати знову до API серверу не розриваючи сесію та потім проаналізувати відповідь. [17]

Наприклад, у запиті до API можна змінити метод GET на метод PUT, що призведе до модифікації даних на API сервері і таким чином здійснити атаку зламаної авторизації на функціональному рівні (Рис. 2.4).

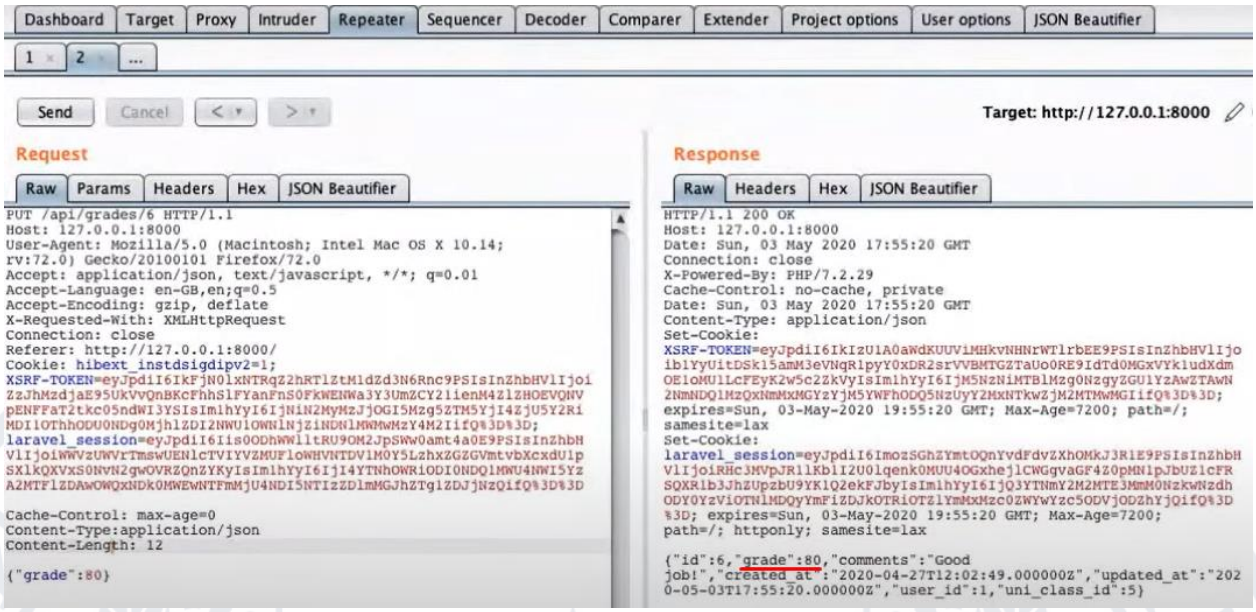


Рис. 2.4. Приклад модифікації даних шляхом зміни HTTP методу запиту з GET на PUT

З рис. 2.4. видно, що запит був змінений таким чином, що замість того, щоб отримати дані про оцінку користувача під номером 6, за допомогою методу PUT ці дані були модифіковані.

Також, схожим способом можна здійснити ескалацію рівня запиту від простого користувача до рівня адміністратора, замінивши у запиті відповідні поля.

У випадку, якщо для авторизації запиту використовується токен, можна дізнатись алгоритм шифрування підпису токена, який вказаний в його заголовку і взламавши підпис, якщо алгоритм шифрування слабкий, змінити в тілі токена дозволи з користувача на адміністратора. Потім, знову ж таки використовуючи інструменти пакету Burp Suite, надіслати HTTP запит до API сервісу із підробленим токеном.

2.3. Тестування обмеження швидкості за допомогою JMeter

JMeter – це програмне забезпечення з відкритим вихідним кодом, призначене для завантаження, тестування продуктивності та поведінки при навантаженні. Спочатку він був розроблений для тестування веб-застосунків, але з тих пір розширився до інших тестових функцій. [18]

Щоб розпочати тестування з допомогою JMeter, необхідно додати у інструменті т. зв. план тестування, до якого входить група потоків та запит, що буде надсилатись при навантаженні. Головними параметрами у групі потоків є кількість потоків, що репрезентують собою користувачів та кількість циклів – тобто кількість повторень (Рис. 2.5). Наприклад, якщо кількість потоків 100, а кількість циклів – 20, то це буде означати, що буде симульовано 100 різних запитів до API 20 разів поспіль.

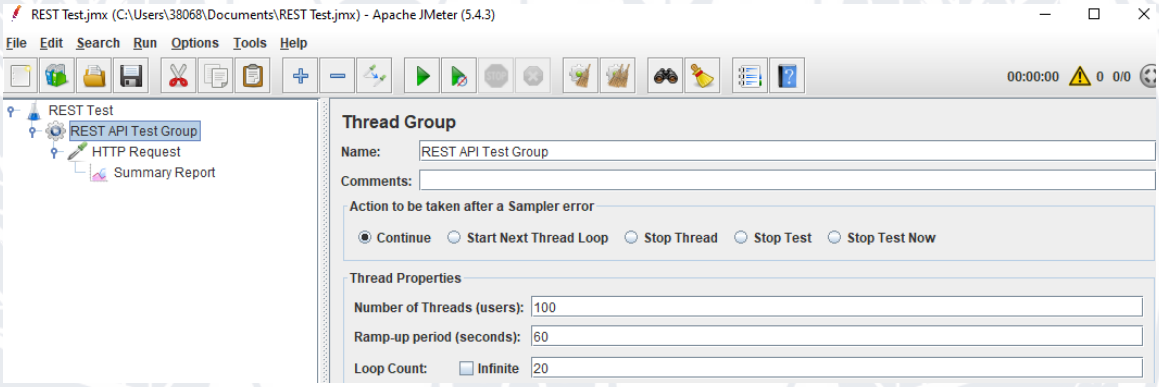


Рис. 2.5. Створення групи потоків REST API Test Group в Apache JMeter

Після створення групи потоків необхідно створити темплейт HTTP запиту, який буде надсилатись до REST API серверу (Рис. 2.6.). Тут основними параметрами є протокол передачі HTTP (HTTP, HTTPS, або FILE), шлях до ресурсу, що тестується та HTTP метод. На рис. 2.6 параметром протоколу є HTTPS а методом є GET. В якості ресурсу було обрано тренувальний веб-ресурс для тестування безпеки з REST API.

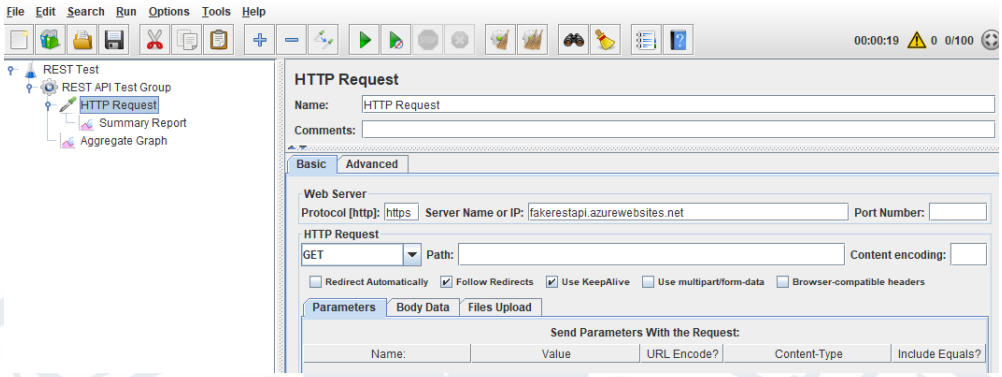
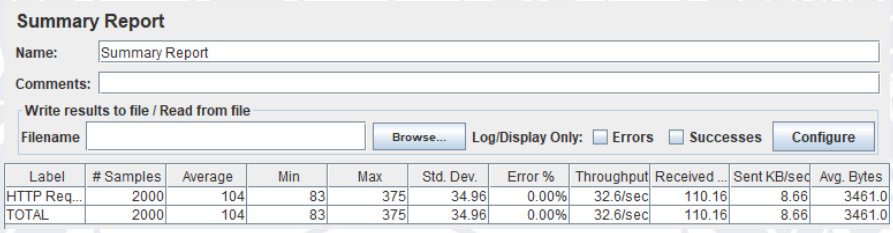


Рис. 2.6. HTTP запит, яким буде тестуватись навантаження

Перед запуском тесту необхідно зберегти план тестування та додати підсумковий звіт. Його можна згенерувати як у вигляді таблиці, так і графіку (Рис. 2.7, 2.8).



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received ...	Sent KB/sec	Avg. Bytes
HTTP Req...	2000	104	83	375	34.96	0.00%	32.6/sec	110.16	8.66	3461.0
TOTAL	2000	104	83	375	34.96	0.00%	32.6/sec	110.16	8.66	3461.0

Рис.2.7 Табличне представлення результатів

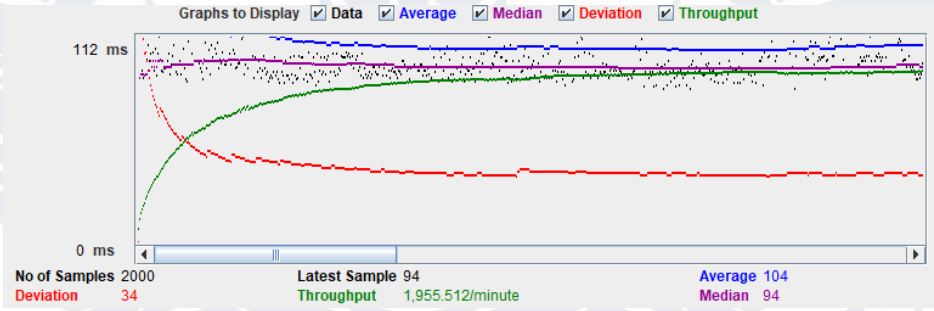


Рис. 2.8. Графічне представлення результатів

З таблиці на рис. 2.7. видно, що усі 100 запитів відпрацювали без помилок. В свою чергу, з графіку, згенерованому на рис. 2.8. видно, що зі збільшенням кількості користувачів росте час затримки (зелена лінія на графіку).

2.4. Аналіз інструментів автоматизованих засобів тестування

У пунктах розділу 2.1 – 2.3 було розглянуто ручне тестування ряду механізмів забезпечення безпеки API сервісів. Зазвичай, такий спосіб набагато надійніший і дозволяє більш ефективно досліджувати вразливості системи. Але з іншої сторони, такий процес потребує величезну кількість фінансових та часових ресурсів. Так, ціна ручного тестування комерційного веб API коливається від 1 тис. доларів до 100 тис. і здійснюється напротязі декількох місяців, в залежності від його глибини. Саме тому значно швидше та дешевше використовувати автоматизовані інструменти тестування веб API. Серед найбільш відомих таких програм є Postman, Katalon Studio і Swagger.

2.4.1. Postman

Postman – це платформа API, яка дозволяє розробникам створювати та тестувати власні API. [19]

Спочатку Postman був плагіном для браузера, розробленим для Google Chrome, тепер також існують версії для Mac та Windows. Postman має зручний графічний інтерфейс та дозволяє створювати автоматичні тести API як без кодування, так і з використанням багатьох різних мов програмування.

Найважливішими особливостями тестування у Postman є:

- Інструменти тестування різних способів автентифікації та авторизації – наприклад, сканер токенів.
- Проксі сканер для перехоплення та аналізу трафіку.
- Тестування стійкості API сервісу за допомогою фазінгових атак (англ. fuzzing) – надсилання надійсних символів у запитах, що може призвести до різного типу помилок.
- Сканування на вразливості частини API, що відповідає за бізнес логіку.

- Інструменти для швидкого та простого написання автоматизованих тестів на різних мовах програмування.
- Можливість створення детальних звітів з описами виявлених вразливостей по закінченню тестування

2.4.2. Katalon Studio

Katalon Studio — це комплексне рішення для автоматизації тестування веб-додатків, API, а також настільних і мобільних додатків. Рішення підтримує запити SOAP і REST, а також широкий спектр параметрів і команд. Katalon Studio та надає інтуїтивний інтерфейс для використання безлічі функцій та команд параметризації для тестування API веб-сервісів для різних платформ, включаючи Windows, Linux та Mac OS.

Katalon Studio підтримує два режими – автоматизоване тестування та ручне тестування. Ручне тестування призначене для досвідчених тестувальників та надає широкий інструментарій для створення тестових сценаріїв для REST та SOAP API. [20]

2.4.3. Swagger

Swagger – це простий у використанні набір інструментів з відкритим кодом для проектування, створення, тестування та документування API. Його особливість в тому, що він надає можливість створювати та тестувати API у реальному часі через спеціальний редактор. [21]

Автоматизація тестування API та перевірка його правильної роботи в різних сценаріях надзвичайно проста за допомогою інструменту ReadyAPI. Користувач можете імпортувати власне API, щоб :

- Автоматично створювати параметризовані запити щодо кінцевих точок API.
- Переглядати роботу запитів та відповідей до API, щоб переконатись, що вони працюють належним чином.
- Здійснювати тестування API під навантаженням.
- Здійснювати детальне документування результатів тестування.

РОЗДІЛ 3. ФОРМУВАННЯ УНІВЕРСАЛЬНИХ МЕТОДИЧНИХ РЕКОМЕНДАЦІЙ З ТЕСТУВАННЯ БЕЗПЕКИ API СЕРВІСІВ

Тестування безпеки API сервісів – це непроста, комплексна задача, у якої немає єдиного рішення. Тестувальники повинні звертати увагу на те, які саме механізми захисту задіяні, як саме здійснюється обмін даними між клієнтом та сервером і як реалізовані API з точки зору бізнес логіки та ін.

Метою виконання тестування безпеки API сервісу є зробити API веб-додатків максимально захищеними, щоб запобігти розголошенню конфіденційної інформації та компрометації веб серверу.

Ціллю тестування є реалізація ряду відомих загроз API сервісів у тестовому середовищі, щоб виявити існуючі вразливостей і у результаті їх аналізу розробити відповідні заходи захисту з ціллю зменшення ризику компрометації API.

У якості основи для створення методології було використано методології тестування, запропоновані у таких фреймворках, як фреймворк оцінювання безпеки інформаційних систем ISSAF (англ. Information System Security Assessment Framework) та посібник з тестування веб-безпеки OWASP WSTG (англ. OWASP Web Security Testing Guide).

ISSAF – це набір методологій, що включає в себе оцінку різних компонент інформаційних систем, починаючи від безпеки хостових систем і закінчуючи оцінкою безпеки веб-додатків. [22]

Посібник з тестування безпеки OWASP є вичерпним посібником з тестування безпеки веб-програм і веб-сервісів. [23]

Процес тестування безпеки будь-якого веб API можна розділити на три послідовних фази (Рис. 3.1.)

- Фаза збору інформації
- Фаза тестування безпеки API
- Аналіз та документування результатів, очищення наслідків

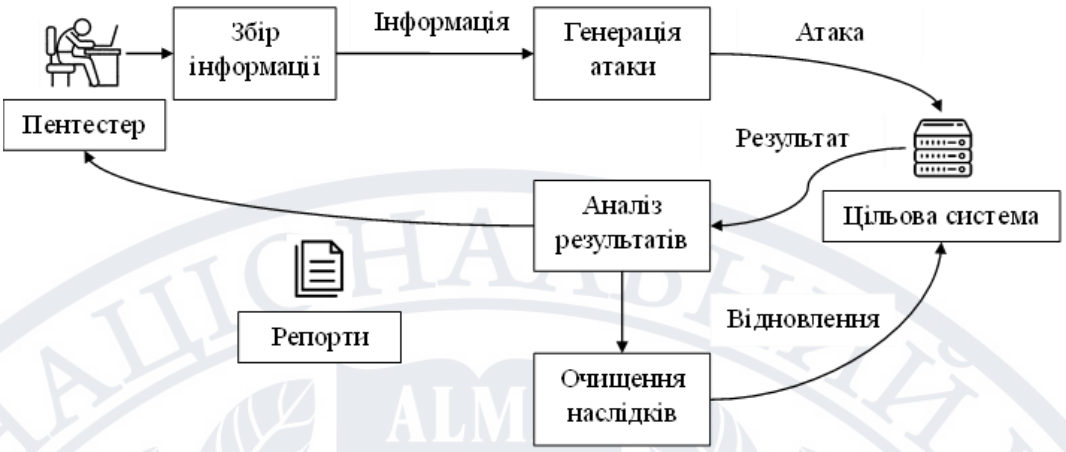


Рис.2.3. Порядок виконання тестування безпеки API сервісів

3.1. Збір інформації

Під час фази збору інформації важливо знайти та проаналізувати усю інформацію про цільове API за допомогою як технічних (прослуховування портів, сканери на вразливості, аналіз структури пакетів запитів та відповідей), так і нетехнічних методів, яку тільки можливо. Це початковий етап аудиту безпеки будь-якої інформаційної системи, який багато людей схильні не помічати. Під час виконання тесту в інформаційній системі збір інформації є надзвичайно важливими та надає всю можливу інформацію, необхідну для продовження процесу тестування. Під час збору інформації важливо проявити якомога більшу уяву.

По закінченню фази збору інформації, пентестер повинен виконати ряд наступних завдань:

- Зрозуміти механізм автентифікації API сервісу та потік даних між клієнтом та API сервером
- Дослідити структуру запитів до API
- Проаналізувати кінцеві точки API

Також тестувальник може використати автоматизоване ПЗ для сканування вразливостей у системі.

3.2. Тестування безпеки API

Мета другої фази тестування – використання інформації, отриманої на протязі першої фази тестування для всебічного тестування безпеки API сервісу та виявлення вразливостей його кінцевих точок.

Головними цілями другої фази є виявлення та експлуатація вразливостей кінцевих точок API за допомогою інструментів як ручного, так і автоматизованого тестування.

Другу фазу тестування в свою чергу, можна поділити на наступні послідовні етапи:

- Тестування механізмів автентифікації та контролю доступу
- Тестування механізмів перевірки вводу
- Тестування механізмів обробки HTTP запитів
- Тестування механізмів обмеження швидкості

Після того, як кінцеву ціль було виконано, можна також протестувати системи безпеки БД, які виявляють зловмисну присутність у системі шляхом реалізації пентестером заходів приховування присутності у системі.

Цього можна досягти такими засобами, як приховані канали, бекдори та руткіти, а також очищення журналу логування.

3.2.1. Тестування механізмів автентифікації та контролю доступу

Автентифікація в REST API реалізовується за допомогою трьох різних способів: базова автентифікація HTTP, API токени та фреймворк OAuth. Кожен із цих способів повинен тестуватись по своєму, оскільки їх імплементація в API сервісах суттєво відрізняється. В залежності від

механізму автентифікації, що використовується у API, можуть бути запропоновані різні методи тестування та відповідний їм інструментарій.

1) Для базової автентифікації HTTP необхідно здійснити тестування наступних елементів механізму безпеки API:

- Тестування передачі даних по захищеному каналу

Тестування передачі даних по захищеному каналу означає впевнення в тому, що дані для автентифікації користувача передаються по зашифрованому каналу для уникнення їх перехоплення.

Щоб оцінити рівень безпеки даного елементу механізму автентифікації, тестувальнику необхідно надсилати GET та/або POST запити і, використовуючи інструмент проксі, такий, наприклад, як Burp Proxy, перехопити та проаналізувати заголовки.

У результаті аналізу необхідно зрозуміти, чи правильно застосовані заходи безпеки передачі використовуючи протокол типу HTTPS, чи використовуються окрім протоколів захищеної передачі даних по каналу додаткові криптографічні алгоритми, наскільки надійними є ключі, які в них використовуються і т. д.

- Тестування облікових даних за замовчуванням

Веб-додатки часто можуть використовувати комерційне ПЗ з відкритим кодом, що спрощує процеси налаштування, необхідні з боку адміністратора. Такі додатки часто після їх встановлення неправильно конфігуровані і облікові дані за замовчуванням, надані для первинної автентифікації, ніколи не змінюються. Використовуючи таку вразливість, можна швидко отримати доступ до записів з такими обліковими даними і здійснивши, наприклад, ескалацію доступу, легко отримати контроль над веб-сервером.

Щоб протестувати API на наявність такої вразливості, тестувальник може здійснити атаку заповнення облікових даних використовуючи інструмент типу Burp Intruder та виконавши дії, описані у п. 2.1.1. розділу 2.

- Тестування механізмів блокування та розблокування

Тестування механізму блокування необхідне, щоб переконатися, що він достатньо ефективний щоб запобігти будь-яким брут-форс атакам підбору паролів. Тестування механізму розблокування необхідне, щоб оцінити його захищеність від спроб неавторизованого розблокування вже заблокованих облікових записів.

В першому випадку тестувальник може використати звичайні брут-форс атаки використовуючи інструменти типу Burp Intruder або ж Hydra.

Відповідно, для оцінки механізму розблокування, необхідно його ініціювати та проаналізувати на вразливості. Зазвичай механізм розблокування включає в себе секретні питання або посилання для розблокування електронною поштою.

- Тестування на можливість обходу схеми автентифікації

Недбалість, незнання або просте заниження загроз безпеці часто призводять до схем аутентифікації, які можна обійти, просто пропустивши сторінку входу та безпосередньо викликавши внутрішню сторінку, доступ до якої передбачається лише після проведення аутентифікації.

Тестування на можливість обходу можливо багатьма різними способами, найвідомішими з яких є т. зв. примусовий браузинг (англ. forced browsing), модифікація параметрів, передбачення ідентифікатору сесії та SQL ін’єкції.

Спосіб примусового браузингу передбачає пряме запитування сторінок, доступ до яких повинен надаватись лише при автентифікації, і, якщо API реалізує контроль доступу лише на сторінці входу, можна отримати доступ до чутливих даних.

В способі модифікації параметрів тестувальник може спробувати змінити фіксовані параметри, що використовуються API в деяких веб сервісах для перевірки успішності автентифікації і таким чином отримати доступ до

захищених сторінок без надання облікових даних. Такі параметри, наприклад, можуть передаватись через URI або в заголовках POST запитів у вигляді кукі.

Передбачення ідентифікатору сесії – це спосіб при якому в API, що використовують легко вгадувані ID сесії, тестувальник може підібрати такий ідентифікатор та таким чином видати себе за іншу особу, що перед цим автентифікувалась у системі.

SQL ін’єкції є досить розповсюдженим способом атак на API. Оскільки у комерційних API вихідний код знаходиться у відкритому доступі можна виконувати вдосконалені атаки проти реалізації процесу аутентифікації. Тестування вразливостей також полегшує широкий наявний інструментарій для проведення атак типу ін’єкцій, наприклад, SQLMap.

3.2.2. Тестування механізмів перевірки вводу

Найпоширенішою слабкістю безпеки API веб-додатків є відсутність належної перевірки введених даних, що надходять від клієнта перед його використанням бізнес логікою сервісу. Ця вразливість веде до міжсайтового скриптингу, ін’єкцій різних типів, атак переповнення буферу та ін. Завданням тестування є застосування різних типів вводу, щоб запевнитись, що API сервісу перевіряє дані вводу перед тим як їх передати на сервер.

Це включає проведення тестів наступних типів:

Тестування з використанням міжсайтового скриптингу (XSS атака) – тестувальник повинен виявити усі можливі неочевидні способи користувацького вводу, наприклад, такі як параметри HTTP, дані POST методу, приховані поля вводу форм та ін., згенерувати спеціальні нешкідливі запити, що дозволять виявити вразливості у отриманих відповідях та на результаті їх аналізу застосувати атаку типу XSS.

Тестування на вразливості до ін’єкцій – тестувальник перевіряє вразливості механізму обробки вводу на наявність можливостей здійснення атак типу SQL, NoSQL, LDAP,ORM, XML та SSL ін’єкцій.

Тестування на вразливості до атак переповнення буферу – тестувальник використовує найбільш розповсюдженні методи тестування переповнення буферу, серед яких: тестування на переповнення кучі (англ. heap overflow), переповнення стеку та тестування вразливості форматування строки.

3.2.3. Тестування механізмів обробки HTTP запитів

REST API HTTP методи грають ключову роль в передачі даних між сервісом та клієнтом. Оскільки таких методів насправді дуже багато, не всі вони необхідні для повноцінної взаємодії між двома сторонами. Деякі з непотрібних методів споживають додаткові ресурси серверу і навіть становлять небезпеку для конфіденційних даних, що на ньому зберігаються. Тому у результаті тестування необхідно запевнитись, що такі HTTP методи, як CONNECT, DELETE, PUT та TRACE не дозволені для користувацького використання на сервері. Ці методи становлять загрозу безпеці, якщо повертають дійсну відповідь, а не помилку.

3.2.4. Тестування механізмів обмеження швидкості

Тестування механізму обмеження швидкості передбачає тестування API системи під навантаженням великої кількості одночасних запитів (по суті, реалізація DoS атаки).

Тестувальник може використати ряд інструментів для реалізації великої кількості одночасних запитів різних HTTP методів, серед яких найефективнішими є Burp Intruder, JMeter, Postman та Docker для того щоб поставити API в умови стресового тесту. Результатом тестування ефективного

механізму протидії атакам перенавантаження та брут-форс є блокування таких спроб від одного і того ж клієнта щодо всіх кінцевих точок API а також відсутність будь-якої інформації про внутрішню структуру API у відповідях з помилками.

3.3. Аналіз та документування результатів, очищення наслідків

Після того, як фазу проникнення було успішно здійснено потрібно проаналізувати та задокументувати результати. Як правило, надані результати включають звіт виконавчого рівня та звіт про технічні висновки. Звіт виконавчого рівня пишеться для потреб керівництва та включає поверхневий огляд оціночної діяльності, її обсягу, найбільш критичних виявлених проблем, загальну оцінку ризиків, сильні сторони безпеки системи та графічна інформація у вигляді знімків екрану. З іншого боку, звіт про технічні висновки повинен включати всі вразливості з деталями щодо того, як відтворити проблему, пов’язані з проблемою ризики активам, рекомендовані дії щодо усунення та корисні довідкові посилання.

Критичним моментом цієї фази є процес відновлення системи до попереднього стану. Усю інформацію, яка була створена та/або зберігалася у системі, слід видалити або приховати, якщо їх усунення через якусь із причин неможливе.

Також ця фаза включає в себе наступні процеси:

- Приховування чи усунення вразливих місць, які вдалось експлуатувати
- Відновлення файлів налаштувань API сервісу, які були змінені під час тестування
- Скасування всіх змін у привілеях та налаштуваннях користувачів
- Видалення із системи всіх інструментів, які використовувались для проникнення

- Документування всіх журналів, зареєстрованих під час тесту, а також видалення їх із тестової системи

ВИСНОВКИ

За результатами роботи можна сформулювати наступні висновки:

1. На основі розгляду REST API як найбільш розповсюдженого представника веб API та його аналізу виділено їх основні загрози та вразливості, серед яких: зламана автентифікація та авторизація, надмірне розкриття даних, ін'єкції, обмеження швидкості, неправильна конфігурація налаштувань безпеки; а також механізми забезпечення безпеки API сервісів: шифрування, автентифікація та авторизація, обмеження швидкості та логування.

2. Запропоновано набір інструментів як ручного, так і автоматизованого тестування заходів безпеки API, наведено практичні приклади їх використання. Серед них:

- Burp Intruder та Burp Repeater як інструменти ручного тестування для створення параметризованого запиту до API системи.
- OAuth.Tools як інструмент тестування механізму автентифікації за допомогою протоколу OAuth 2.0.
- JWT-cracker як інструмент зламу та підробки API токенів.
- JMeter як інструмент створення штучного навантаження.
- Postman, Katalon Studio та Swagger як інструменти для сканування на вразливості та здійснення автоматизованого тестування.

3. Сформовано методичні рекомендації по проведенню тестування безпеки API на основі існуючих методологій тестування безпеки інформаційних систем, таких, як OWASP Web Security Testing Guide та ISSAF. В створених рекомендаціях було запропоновано наступні фази тестування:

1) Фаза збору інформації, метою якої є збір інформації про цільове API та сканування його на вразливості.

2) Фаза тестування механізмів безпеки API, цілями якої є виявлення та експлуатація вразливостей кінцевих точок API за допомогою інструментів

як ручного, так і автоматизованого тестування, яку можна поділити на наступні послідовні етапи тестування:

- Тестування механізмів автентифікації та контролю доступу
- Тестування механізмів перевірки вводу
- Тестування механізмів обробки HTTP запитів
- Тестування механізмів обмеження швидкості

3) Фаза аналізу результатів та очищення наслідків, під час якої відбувається документування та відновлення системи до передтестового стану.

СПИСОК ВИКОРИСТАННИХ ДЖЕРЕЛ

1. Reddy, M. API Design for C++. 2011. 472 p.
2. Lauret, A. The Design of Web APIs. New York, 2019. 392 p.
3. Jacobson, D. APIs: A Strategy Guide. 2011. 149 p.
4. What is API: Definition, Types, Specifications, Documentation [Електронний ресурс]. Режим доступу: https://www.altexsoft.com/blog/engineering/what-is-api-definition-types-specifications-documentation/ (дата звернення: 15.04.2022)
5. Masse, M. REST API Design Rulebook. 2012. 112 p.
6. Shuler, R. How Does the Internet Work? 2002 [Електронний ресурс]. Режим доступу: https://web.stanford.edu/class/msande91si/www-spr04/readings/week1/InternetWhitepaper.htm (Дата звернення: 15.04.2022)
7. An overview of HTTP [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview#what_can_be_controlled_by_http (Дата звернення: 15.04.2022)
8. Webber, J. REST In Practice. 2010. 448 p.
9. Fielding, R. Representation State Transfer. 2000. 180 p.
10. Madden N. API Security in Action. 2020. 576 p.
11. OWASP API Security Project. 2019. [Електронний ресурс] Режим доступу: https://owasp.org/www-project-api-security/ (Дата звернення: 22.04.2022)
12. Mueller, N. Credential Stuffing. 2021. [Електронний ресурс] Режим доступу: https://owasp.org/www-community/attacks/Credential_stuffing (Дата звернення: 22.04.2022)
13. Levin, G. Top 7 REST API Security Threats. 2019. [Електронний ресурс] Режим доступу: https://blog.restcase.com/top-7-rest-api-security-threats/ (Дата звернення: 22.04.2022)

14. Lakshmiraghavan B. Pro ASP.NET Web API Security: Securing ASP.NET Web API. 2013. 433 p.
15. Port Swigger. Burp Intruder. 2022. [Електронний ресурс] Режим доступу: https://portswigger.net/burp/documentation/desktop/tools/intruder (Дата звернення: 24.04.2022)
16. Jason Web Token. 2018. [Електронний ресурс] Режим доступу: https://jwt.io/ (Дата звернення: 27.04.2022)
17. Using Burp Repeater. 2022. [Електронний ресурс] Режим доступу: https://portswigger.net/burp/documentation/desktop/tools/repeater/using (Дата звернення: 27.04.2022)
18. JMeter. 2018 [Електронний ресурс] Режим доступу: https://jmeter.apache.org/ (Дата звернення: 28.04.2022)
19. Postman. [Електронний ресурс] Режим доступу: https://learning.postman.com/docs/ (Дата звернення: 28.04.2022)
20. Katalon Studio. 2020. [Електронний ресурс] Режим доступу: https://docs.katalon.com/katalon-studio/docs/index.html (Дата звернення: 28.04.2022)
21. Swagger. API Testing. [Електронний ресурс] Режим доступу: https://swagger.io/solutions/api-testing/html (Дата звернення: 28.04.2022)
22. ISSAF. Information Systems Security Assessment Framework. 2006. 1251 p.
23. OWASP. OWASP Web Security Testing Guide 4.0. 2014. 224 p.