Софтуерната сигурност е защита срещу: загуба на данни, прекъсване на услуга, изтичане и/или несъответствие на данните.

* Загуба на данни – основните данни се губят поради нарушение на сигурността, т.е. нападателят получава достъп до основната база данни и изтрива записите. Пример: [Hacker deleted all data from VFEmail Servers, including backupsSecurity Affairs](https://securityaffairs.co/81030/hacking/vfemail-destructive-cyberattack.html)
* Прекъсване на услугата - Дейността на системата е нарушена поради действия на атакуващия, т.е. нападателите организират атака, водеща до отказ на услугите на системата. Пример: 
* Изтичане на данни - Основните данни се крадат и се предоставят на неоторизирани получатели, т.е. нападателите получават достъп до базата данни и крадат важна информация, като например кредитни карти. Пример: [Marriott Hackers Stole Data On 500 Million Guests -- Passports And Credit Card Info Included (forbes.com)](https://www.forbes.com/sites/thomasbrewster/2018/11/30/marriott-admits-hackers-stole-data-on-500-million-guests/?sh=50f10ba46492#786737086492) 
* Несъответствие на данните - данните се манипулират от неоторизирани нападатели и стават непоследователни, т.е. нападателите се представят за някой друг и извършват неоторизирани действия.

Сигурността на софтуера включва терминология, използваща се в много дискусии. Таблица х.х. описва някои основи термина.

|  |  |  |
| --- | --- | --- |
| Име | Описание | Пример |
| Заплаха (Threat) | Събитие, което, ако се случи, ще доведе до инцидент със сигурността. | SQL Injection, DDOS атака |
| Атака (Attack) | Действително изпълнение на заплаха от нападател(и). | Злонамерен потребител въвежда SQL Injection или група нападатели организират DDOS атака. |
| Уязвимост (Vulnerability) | Проблем в системата, който може да бъде използван от атакуващия, например за компрометиране на данните. | Неправилно конфигурираната защитна стена излагаща вътрешните системи. |
| Удостоверяване (Authentication) | Установяване на самоличността на потребител (човек или ядошкеи) въз основа на надеждни  Механизми. | потребителско име / парола, СМС или биометрична идентификация |
| Упълномощаване (Authorization) | Установяване какво е разрешено да прави даден потребител в системата. | Потребителят X има право да създаде нова поръчка. Потребителят X НЯМА право да отмени съществуваща поръчка. |

Процес на защитена архитектура гарантира, че системата е възможно най-добре защитена. Той включва всички фази на системата, ръководен от мениджъра на проекта и архитекта на системата. Процесът се състои от 5 етапа:

* Моделиране на заплахи - Процес за идентифициране на потенциални заплахи за система. Включва почти всички от софтуерния екип, като може да използва официални методи и инструменти. Процесът по моделиране включва три основни въпроса: 1. Какво се изгражда? 2. Какво може да се обърка? 3. Как може да се предотврати това?
* Сигурна архитектура – интегрира защитата на сигурността в ядрото на архитектура. Основава се на концепцията за периметър на сигурност. Отнася се за граница, която разделя вътрешната мрежа, системите и данните на организацията от външните мрежи като интернет. Традиционните мерки за сигурност като защитни стени, системи за откриване на проникване и антивирусен софтуер често се разполагат на ръба на тази граница, за да защитят вътрешната среда от външни заплахи. Въпреки това, еволюция в технологиите създава динамичен цифров „периметърът“, който не е фиксирана линия, а по-скоро постоянно променяща се граница.
* Тази промяна доведе до нови парадигми, включително:   
  Zero Trust Network Architecture и Secure Access Service Edge (SASE)
* Мрежова архитектура с нулево доверие: Този модел работи на предпоставката, че заплахите могат да идват както отвън, така и отвътре в периметъра на сигурността, следователно „не се доверявайте на никого“ се превръща във водещ принцип. Zero Trust налага стриктна проверка на самоличността на всеки човек и устройство, които се опитват да получат достъп до ресурси в частна мрежа, независимо дали се намират в мрежата или извън нея.
* Secure Access Service Edge (SASE): Този модел съчетава мрежова сигурност и възможности за широкообхватна мрежа (WAN) в една облачна услуга, за да позволи сигурна и бърза облачна трансформация. Той опростява управлението и внедряването на мрежова сигурност, като предоставя и двете като унифицирана услуга.
* Жизненият цикъл на защитена разработка (SDLC) - подход за изграждане на приложения и софтуерни системи, който включва практики и принципи за сигурност от самото начало и през всички фази на процеса на разработка. SDLC може да се разглежда като интеграция на традиционни процеси на разработка с практики за сигурност, за да се гарантира, че уязвимостите в сигурността се идентифицират и адресират възможно най-рано. Общата цел на SDLC е да намали риска от уязвимости на сигурността в крайния продукт чрез интегриране на сигурността в процеса на разработка, вместо да се опитва да я добави по-късно като последваща мисъл. Това е по-проактивен подход към сигурността и има тенденция да води до по-сигурни продукти, тъй като третира сигурността като основен компонент на разработката на системата и софтуера.
* Ориентирано към сигурността тестване - включва различни видове тестване, насочени към потенциални уязвимости. Някои от тях са статичен и динамичен анализ. Статичния анализ включва анализиране на код без неговото изпълнение, като се използват автоматизирани инструменти за намиране на потенциални уязвимости, свързани с несигурни практики за кодиране. Динамичен анализ е тип тестване включва анализиране на приложение, докато работи. Това може да включва тестване на откритите интерфейси на приложението за уязвимости (fuzz testing) или тестване на системата за уязвимости, които се появяват по време на работа.   
  Друг вид тестване е реален тест за сигурността на системата, при който етични хакери се опитват да използват уязвимостите във вашата система по контролиран начин. След извършване на тестването, трябва да се анализират резултатите, като същевременно се идентифицират всички уязвимости, открити по време на процеса на тестване. След това трябва да се приоритизират тези уязвимости въз основа на фактори като потенциално въздействие, лекотата, с която могат да бъдат използвани, и критичността на системния компонент, където са открити. Стратегии за справяне с тези уязвимости, може да включват пренаписване на код, промяна на системните конфигурации или внедряване на нови контроли за сигурност.

След анализиране резултатите от тестовете, трябва да се сравнят с модела на заплаха, включвайки: проверка дали откритите по време на тестването уязвимости са включени във модела, проверка дали най-сериозните уязвимости съответстват на най-вероятните заплахи, дали определени типове заплахи от модела не са идентифицирани при тестване. Целта е не само да се открият и коригират уязвимостите, но и да се подобри цялостното разбиране за сигурността на системата.

Еволюцията на разработването на приложения и необходимостта от сигурни методи за удостоверяване, които да съответстват на променящите се технологии. В резюме, може да отбележим няколко етапа:

В миналото приложенията бяха предимно дебели клиентски приложения (като Windows Forms или WPF приложения) или сървърни уеб приложения, като и двете не бяха базирани на услуги. Тези приложения често използваха идентификационни данни на Windows или удостоверяване на формуляри и тяхната сигурност беше сравнително ясна поради изолацията им в рамките на един домейн или интранет на компанията.

Пейзажът започна да се променя с възхода на ориентирани към услуги приложения, често направени с помощта на рамки като Windows Communication Foundation (WCF). Въпреки това, това все още са обикновено дебели клиентски приложения в рамките на фирмен домейн или комуникации между сървъри, като WCF се справя с повечето проблеми със сигурността.

Тъй като ландшафтът за разработка на приложения продължи да се развива, ние започнахме да изграждаме уеб API и RESTful API, които не винаги се намират в същия домейн или под наш контрол. Приложенията започнаха да се интегрират с различни API, включително услуги на трети страни като Google Maps.

С преминаването към уеб приложения от страна на клиента, мобилни приложения и API, комуникиращи с други API, традиционните мерки за сигурност като поддържане на API в рамките на стените на компанията или използване на удостоверяване на формуляри станаха недостатъчни. Не можем да защитим нашите API, като просто ги запазим вътрешни, особено за мобилни или клиентски приложения, които трябва да взаимодействат с публичен уеб API.

В отговор разработчиците започнаха да използват система за сигурност, базирана на токени. Това включва изпращане на токени вместо комбинации от потребителско име и парола с всяка заявка за API. Токените представляват потребителско съгласие и могат да бъдат предадени на API за разрешаване на достъп. Създаването и безопасното доставяне на тези токени обаче повдигна нови предизвикателства.

Първоначално решение беше да се изградят услуги за токени, приемащи потребителско име и парола и връщане на JSON Web Token (JWT). Този метод е подобрение, но все още изисква споделяне на поверителна информация с приложението или API и включва много персонализирано внедряване.

Пасажът завършва с идентифициране на необходимостта от централен доставчик на идентичност, който може да се справи с общи задачи, свързани с идентичността, и доказан протокол, който се справя с общи проблеми, които може да пренебрегнем, когато сами създаваме услуги за токени.

Преди това приложенията отговаряха за собственото си удостоверяване на потребителя, включително екрани за влизане, управление на потребителите, правила за пароли и т.н. Днес този подход е по-малко предпочитан, защото децентрализира и усложнява сигурността.

Ролята за удостоверяване на потребител сега обикновено се управлява от централизиран доставчик на идентичност (IdP). Тези IdP често се справят и със задачи за управление на акаунти. Чрез централизирането на тези задачи системата става по-управляема, сигурна и многократно използваема в различни приложения.

Има няколко причини да предпочетете централизирани системи за управление на идентичността и достъпа. Те намаляват вероятността от грешки, по-лесни са за актуализиране и тестване и могат да управляват набор от задачи от регистрация на потребител до заключване на акаунт.

Пасажът дава пример за сигурност с парола, за да илюстрира необходимостта от централизация. Безопасното съхраняване на пароли еволюира от основно криптиране до методи за разтягане на ключове като Argon2 или bcrypt. Най-добрата практика е тази функция да се централизира, така че когато стандартите се развиват, те да могат да се актуализират на едно място.

В допълнение към обработката на пароли, IdPs могат да управляват други средства за удостоверяване, като сертификати или удостоверяване от втори и трети фактор. По-ефективно е тези опции да се управляват централно, отколкото да се налага да актуализирате всяко клиентско приложение поотделно.

Преминаването към централизация предоставя решения на първия проблем: преместване на процеса на удостоверяване на потребителя от отделни приложения към ниво IdP.

След като е налице централизиран IdP, на това ниво може да се добави код за обработка на крайни точки за влизане или токени. Това обаче не решава напълно проблемите, свързани с местните токен услуги, тъй като все още може да доведе до преоткриване на колелото.

OAuth 2 и OpenID Connect, предоставят протоколи за справяне с общи проблеми и рационализират процеса на създаване и доставка на токени.