Bezpieczeństwo w inżynierii oprogramowania

Architektura oprogramowania a bezpieczeństwo

Arkadiusz Chrobot

Katedra Systemów Informatycznych

16 listopada 2024

Plan

Wstęp

Architektura oprogramowania ma bezpośrednie przełożenie na to, czy będzie ono spełniało wymagania niefunkcjonalne, do których zalicza się bezpieczeństwo (ang. *security*) [5]. Istnieje wiele podobnych pojęć, jak *Architektura systemowa* i *architektura cyberbezpieczeństwa*, dlatego pierwsza część wykładu poświęcona jest definicjom zagadnień, które będą omawiane. Druga część opisuje zasady, które można zastosować w projektowaniu, aby uzyskać bezpieczną architekturę oprogramowania. W trzeciej części przedstawiono wynikające z tych zasad taktyki zabezpieczania systemów oprogramowania, a w czwartej przykładowe wzorce architektoniczne, które wspierają bezpieczeństwo.

Definicje

Architektura bezpieczeństwa

Architektura bezpieczeństwa (ang. *security architecture*) lub architektura cyberbezpieczeństwa (ang. *cybersecurity architecture*) może być zdefiniowana jako ogół środków i relacji między nimi, które są zastosowane, aby zapewnić bezpieczeństwo aktywów i zasobów informatycznych. Swoim zakresem nie tylko obejmuje zabezpieczenia stricte informatyczne, ale również fizyczne, takie jak zamki, klucze, drzwi. [6].

Architektura systemu

Architektura systemu (ang. *system architecture*) może być zdefiniowana jako ogół komponentów i relacji między nimi, które składają się na całość systemu informatycznego. Swoim zakresem może obejmować nie tylko oprogramowanie, ale również strukturę sieci komputerowej oraz stosowane platformy sprzętowe.

Definicje

Architektura oprogramowania

W zasadzie nie istnieje jedna uniwersalna definicja architektury oprogramowania. Najbliższa ideału jest definicja podana przez pracowników CMU SEI: architektura oprogramowania (ang. *software architecture*) reprezentuje decyzje projektowe, związane z jego ogólną strukturą i zachowaniem.

Architektura oprogramowania przedstawia *strukturę* oprogramowania z czterech perspektyw [1]:

- **statycznej** definiującej poszczególne komponenty,
- **dynamicznej** określającej podział na procesy i/lub wątki w trakcie wykonania,
- **interfejsów** specyfikującej usługi dostarczane przez komponenty za pomocą ich publicznych interfejsów,
- **zależności** określającej przepływ danych i inne relacje między komponentami.

Zasady projektowania

Zasady, którymi należy się kierować chcąc zaprojektować bezpieczną architekturę oprogramowania, zostały podane w artykule Jerome'a Saltzera i Michaela Schroedera, dwóch naukowców z MIT, z 1975 roku [4]:

- prostota projektu,
- przejrzystość projektu,
- zasada najmniejszych uprawnień,
- zasada najmniejszej ilości informacji,
- bezpieczeństwo z założenia (ang. *secure by default*),
- listy dozwolonych nad listy zabronionych,
- unikanie przewidywalności,

Zasady projektowania

- bezpieczne awarie (ang. *fail safe*),
- pełna mediacja,
- jak najmniej wspólnych mechanizmów,
- nadmiarowość,
- głęboka obrona (ang. *defense in depth*),
- rozdzielanie uprawnień (ang. *separation of privileges*),
- zasada ograniczonego zaufania.

Prostota projektu

Parafrazując powiedzenie Alberta Einsteina można stwierdzić, że zgodnie z tą regułą projekt oprogramowania powinien być prosty, ale nie prostszy. Wymaga ona, aby w architekturze stosować możliwie proste komponenty realizujące pojedyncze funkcje, ale współpracujące z innymi komponentami. W kontekście bezpieczeństwa te funkcje mogą obejmować np. uwierzytelnianie i autoryzację. Należy zwrócić uwagę, że ta zasada nie wyklucza użycia bardziej złożonych komponentów tam, gdzie jest to uzasadnione.

Przejrzystość projektu

Ta reguła wywodzi się z zasady sformułowanej przez dziewiętnastowiecznego holenderskiego kryptografa Auguste'a Kerckhoffs'a, która głosi, że bezpieczeństwo kryptosystemu musi być oparte na jak najmniejszej ilości tajnej informacji, czyli tylko kluczu, który powinien być dodatkowo łatwy do zmiany. Podobnie, architektura oprogramowania powinna być jawna, jej szczegóły powinny być znane nawet potencjalnym intruzom. Bezpieczeństwo powinno wynikać z tego jak dobrze jest ona zaprojektowana, a nie z tajności jej szczegółów. Przeciwieństwem tej reguły jest podejście nazywane bezpieczeństwem przez zagmatwanie/zaciemnienie (ang. *security by obscurity*).

Zasada najmniejszych uprawnień

Zasada najmniejszych uprawnień (ang. *least privilege*), nazywana także zasadą *wiedzy koniecznej* (ang. *need-to-know principle*) wymaga, aby każdy komponent architektury (np. proces, wątek) miał dostęp tylko do tych zasobów, które *w danej chwili* są konieczne do wykonania jego pracy. Jeśli dane uprawniania przestaną mu być potrzebne powinny mu być jak najszybciej odebrane. Przyznanie nowych uprawnień powinno wiązać się z weryfikacją jego tożsamości. W ten sposób, nawet jeśli intruz przejmie jakiś komponent, to będzie miał ograniczone możliwości wyrządzenia szkód. Ta zasada zapobiega także powstaniu szkody incydentalnej.

Zasada najmniejszej ilości informacji

Przepływ danych między komponentami powinien być ograniczony tylko do niezbędnego minimum. Oprogramowanie jako całość nie powinno gromadzić danych, które nie są niezbędne (np. danych osobowych). Zasada ta jest trudna do przestrzegania z uwagi na ewolucję interfejsów komponentów, która może skutkować koniecznością wykorzystywania tych samych parametrów lub struktur danych. Należy też zwrócić uwagę na informacje przekazywane w komunikatach, w szczególności tych, które dotyczą wystąpienia błędów. Bufory danych powinny być, zgodnie z tą regułą, opróżnione najszybciej, jak to jest możliwe.

Bezpieczeństwo z założenia

Zasada bezpieczeństwa z założenia, nazywana również zasadą domyślnego bezpieczeństwa (ang. *safe by default*). Wymaga ona, aby oprogramowanie miało konfigurację, która zawsze będzie gwarantowała jego bezpieczeństwo, nawet zaraz po wdrożeniu. Przykładem łamania tej zasady są domyślne hasła. Można wymusić ich zmianę poprzez wyłączenie pełnej funkcjonalności oprogramowania (np. nie połączy się ono z Internetem) do chwili ich modyfikacji. W przypadku np. urządzeń IoT stosowana jest metoda „zmartwychwstającego kaczątka” (ang. *resurrecting duckling*), tzn. urządzenie z domyślną konfiguracją (np. powstałą po przywróceniu ustawień domyślnych) łączy się wyłącznie z najbliższym punktem dostępowym.

Listy dozwolonych nad listy zabronionych

Zasada ta ma wiele zastosowań, ale główne związana jest z weryfikacją danych. Zazwyczaj łatwiej jest na przykład określić jakie znaki są dozwolone, niż wyznaczyć zbiór tych, które muszą być odrzucone. Stąd zaleca się częściej stosować podejście „białej listy” (ang. *white list, allowed list*), niż czarnej (ang. *black list, blocked list*). Użycie tego ostatniego nie jest jednak przez tę regułę wykluczone, bo sprawdza się w niektórych dziedzinach. Przykładem są programy antywirusowe, których działanie bazuje na szukaniu sygnatur złośliwego kodu. W tym przypadku łatwiej jest wskazać, które programy są niedozwolone, niż te, które są dopuszczone.

Unikanie przewidywalności

Należy unikać tworzenia łatwych do odgadnięcia identyfikatorów zasobów lub takich (np. kolejne liczb naturalne) lub takich, których ujawnienie byłoby równoznaczne z wyciekiem danych (np. tworzonych na podstawie imienia, nazwiska i numeru telefonu użytkownika). Zaleca się stosowanie bezpiecznych kryptograficznie generatorów liczb pseudolosowych do uzyskiwania danych identyfikujących.

Bezpieczne awarie

Zasada bezpiecznej awarii (ang. *fail safe*) wymaga, aby każdy komponent zachowywał bezpieczny stan nawet w przypadku wystąpienia poważnych problemów. Wdrożenie tej zasady wymaga dokładnej implementacji obsługi wyjątków oraz starannego jej testowania. Nieścisłości w obsłudze sytuacji wyjątkowych są często wykorzystywane przez intruzów do uzyskania nieuprawnionego dostępu do zasobów oprogramowania.

Pełna mediacja

Pełna mediacja (ang. *complete mediation*) oznacza spójne i bezpieczne weryfikowanie wszystkich ścieżek dostępu do chronionego zasobu. Przykładowo, wszystkie żądania do bazy danych powinny być walidowane przez jeden komponent, który będzie sprawdzał, czy nie zawierają one danych wskazujących na atak typu *wstrzyknięcie SQL* i w razie konieczności będzie te dane usuwał, neutralizował lub blokował. Taki komponent zwykle nazywany jest *strażnikiem*, ale może także stać się wąskim gardłem (ang. *bottleneck*) oprogramowania. Z tego względu czasem zastosowanie takiego podejścia nie jest możliwe i w zamian stosuje się wiele komponentów-strażników, ale funkcjonalnie równoważnych.

Pełna mediacja

Ogólnie, można wskazać trzy poziomy zgodności z tą zasadą:

wysoka zgodność dostęp do zasobu jest możliwy tylko przez jeden komponent;

średnia zgodność dostęp do zasobu jest możliwy przez wiele komponentów, ale każdy z nich przeprowadza weryfikację dostępu w ten sam sposób;

niska zgodność dostęp do zasobu jest możliwy przez wiele komponentów, z których każdy inaczej weryfikuje żądanie (*niepełna mediacja*).

Jak najmniej wspólnych mechanizmów

Ta zasada wymaga, aby niezależne procesy miały jak najmniej wspólnych mechanizmów, z których działania korzystają. Im więcej jest takich mechanizmów, tym większe prawdopodobieństwo, że dojdzie do zmostkowania (ang. *bridge*) komponentów i komponent mniej uprzywilejowany otrzyma dostęp do danych komponentu bardziej uprzywilejowanego. Ta zasada jest stosowana m.in. w systemach operacyjnych (izolacja procesów użytkowników), bazach danych (osobne schematy i przestrzenie nazw dla poszczególnych użytkowników) lub w aplikacjach internetowych (każdy klient przechowuje przeznaczone dla niego i tylko niego ciasteczka).

Nadmiarowość

Stosowanie komponentów nadmiarowych (ang. *redundant*) ma miejsce głównie w systemach o krytycznych zastosowaniach (ang. *safety--critical*). Usługi tych systemów muszą charakteryzować się wysoką dostępnością (ang. *high availability*) i niezawodnością. Te własności są także ważne w systemach bezpiecznych. Komponenty nadmiarowe mogą pracować wspólnie z komponentami podstawowymi, a ich wyniki mogą podlegać głosowaniu większościowemu lub stanowić tzw. zimny zapas.

Głęboka obrona

Reguła głębokiej obrony (ang. *defense in depth*), nazywana także regułą obrony wielowarstwowej zakłada, że dostęp do aktywów jest chroniony za pomocą kilku warstw mechanizmów zabezpieczających, a nie tylko jednej. Przykładem może być ochrona przed atakami typu *SQL Injection*, która może polegać na walidacji danych po stronie klienta (zawodnej i niezaufanej), serwera (bardziej bezpiecznej i zaufanej), a nawet systemu zarządzania bazą danych.

Rozdzielanie uprawnień

Ta reguła wymaga, aby uprawnienia mające krytyczne znaczenie dla bezpieczeństwa oprogramowania, były rozdzielone między co najmniej dwa komponenty. Przykładowo, w systemach bankowości elektronicznej nawet uwierzytelniony użytkownik musi osobno autoryzować każdą transakcję, a w przypadku firm przelewy na znaczne kwoty wymagają autoryzacji przez więcej niż jedną osobę (np. przez główną księgową i dyrektora finansowego). Ściśle informatycznym zastosowaniem tej reguły jest uwierzytelnianie wieloskładnikowe. Tożsamość użytkownika jest weryfikowana na bazie więcej niż jednej danej.

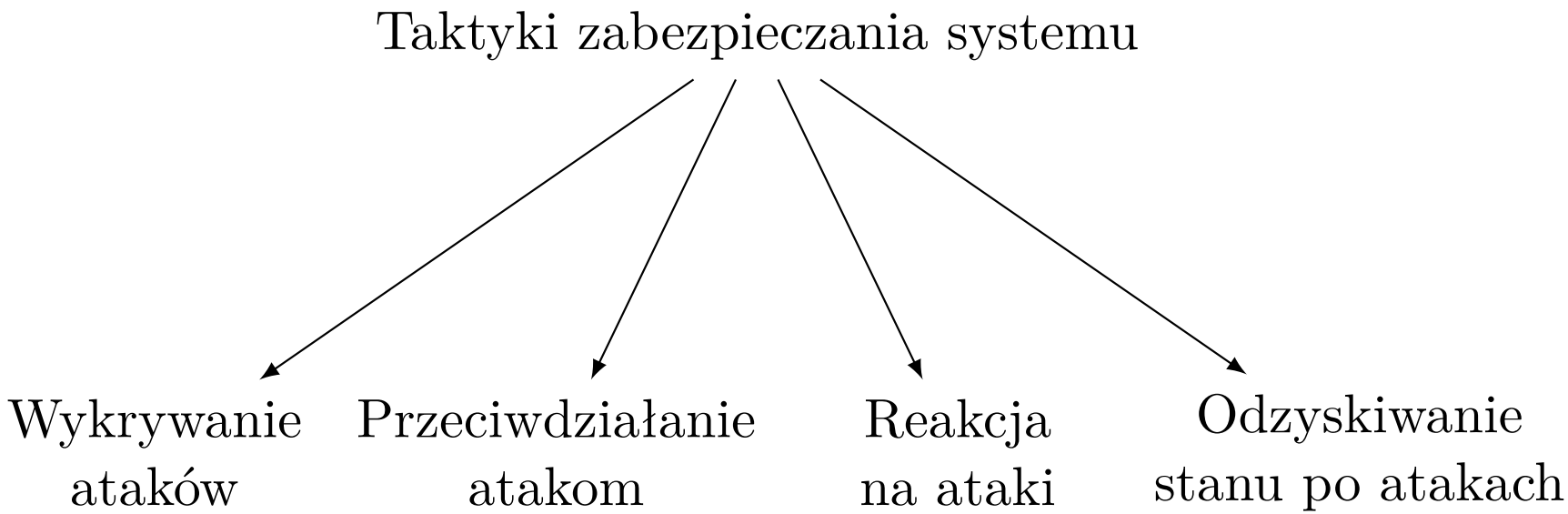
Zasada ograniczonego zaufania

Przykładem zastosowania tej metody jest koncepcja *zero zaufania* (ang. *zero trust*), według której żaden komponent nie powinie ufać innemu, ale przed spełnieniem jego żądania lub przetworzeniem danych od niego otrzymanych powinien potwierdzić jego tożsamość. Zasada ta może kolidować z wymaganiami wydajności, ale jej spełnienie może być konieczne w przypadku projektowania systemów bezpiecznych.

Taktyki zabezpieczania

Z omówionych reguł wynikają *taktyki zabezpieczania*, które pozwalają radzić sobie z typowymi problemami związanymi z bezpieczeństwem. Dzieli się je na cztery główne grupy (Rysunek 1). Każda z tych kategorii zawiera po kilka taktyk [2].

Taktyki zabezpieczania



Kategorie taktyk zabezpieczania

Taktyki wykrywania ataków

Do tych taktyk zalicza się *wykrywanie włamania*, które polega na monitorowaniu ruchu sieciowego i poszukiwaniu w nim sygnatur wskazujących na próby ataku lub atak w trakcie realizacji (np. pakiety o nietypowo dużych rozmiarach, pakiety z nietypowych adresów, itp.). Inną taktyką tego typu jest *wykrywanie niedostępności usługi*, która sprowadza się do szukania w ruchu sieciowym oznak ataku DoS lub DDoS. Do tej samej kategorii zaliczane jest *weryfikowanie integralności plików i komunikatów*, polegające na stosowaniu i sprawdzaniu sum kontrolnych danych, które są przesyłane lub przechowywane przez oprogramowania. Ostatnią taktyką z omawianego rodzaju jest *wykrywanie anomalii w dostarczaniu komunikatów*, polegająca na badaniu czasu przesyłania informacji, a w szczególności jego wariancji. Dzięki niej można wykryć ataki typu *man-in-the-middle*.

Taktyki przeciwdziałania atakom

Do tej kategorii taktyk zalicza się:

identyfikowanie jednostek czyli sprawdzanie tożsamości potencjalnych źródeł danych dla oprogramowania,

uwierzytelnianie jednostek czyli potwierdzanie tej tożsamości, albo określonych cech jednostek (np. sprawdzanie za pomocą CAPTCHA, czy użytkownik jest człowiekiem),

autoryzacja jednostek czyli ustalenie, czy jednostce przysługuje dostęp do zasobu i na czym on może polegać,

ograniczenie dostępu czyli zmniejszenie powierzchni ataku poprzez minimalizację liczby punktów dostępu,

ograniczenie narażenia na atak czyli rodzaj ochrony pasywnej minimalizującej skutki potencjalnego ataku,

szyfowanie danych dotyczące zarówno przechowywanych, jak i przesyłanych informacji,

Taktyki przeciwdziałania atakom

rodzielenie jednostek polega na odseparowaniu jednostek z użyciem maszyn wirtualnych, kontenerów, „szczelin powietrznych” lub „galwanizacji” (ang. *air gap*), aby zmniejszyć stopień ich narażenia na ataki,

walidacja danych wejściowych czyli oczyszczanie danych z potencjalnie niebezpiecznej zawartości.

zmiana domyślnych ustawień związanych z uwierzytelnieniem czyli wymuszenie zmiany domyślnych haseł.

Taktyki reakcji na ataki

Ta kategoria obejmuje:

odebranie dostępu oprogramowanie powinno posiadać mechanizmy, które w obliczu ataku pozwolą ograniczyć dostęp do zasobów, nawet użytkownikom normalnie do tego uprawnionym,

ograniczenie możliwości logowania w przypadku wykrycia wielokrotnych nieudanych prób logowania, konto którego te próby dotyczą może zostać zablokowane w sposób trwały lub czasowy,

informowanie jednostek o wykryciu ataku powinny być poinformowane odpowiednie jednostki (ludzie lub systemy), mające za zadanie mu przeciwdziałać.

Taktyki odzyskiwania stanu po ataku

W tej kategorii, oprócz taktyk związanych z zapewnianiem dostępności, znajdują się:

audyt czyli zapewnienie możliwości śledzenia i badania ataku poprzez rejestrowanie działań użytkowników oprogramowania,

niezaprzeczalność czyli rejestrowanie danych, które uniemożliwią sprawcy wyparcia się uczestniczenia w ataku.

Wzorce architektoniczne

Wzorce architektoniczne są gotowymi schematami, które można stosować do rozwiązywania znanych problemów w projektowaniu [3]. W przypadku bezpieczeństwa dwa najpopularniejsze to:

- **Repozytorium**, nazywany także wzorcem *współdzielonych danych* [3] lub *centralnej bazy danych* [1]. W tym wzorcu dane są przechowywane w repozytorium, które udostępnia je innym komponentom. Dzięki zgromadzeniu ich w jednym miejscu łatwiej jest zarządzać dostępem do nich, tworzyć ich kopie zapasowe, itd.
- **Architektura wielowarstwowa** nazywany także *architekturą warstwową* lub *architekturą abstrakcyjnych maszyn* [1]. W tym wzorcu każdy komponent jest warstwą, która korzysta z usług warstwy znajdującej się „pod nią” w modelu i sama dostarcza usług warstwie znajdującej się „nad nią”. Aktywa umieszcza się w najbardziej wewnętrznej chronionej warstwie. Przejścia żądań między warstwami są starannie weryfikowane.

Źródła I


Ian Sommerville. *Inżynieria oprogramowania*. PWN, Warszawa, 2020.


Len Bass, Paul Clements i Rick Kazman. *Architektura oprogramowania w praktyce*. Helion, Gliwice, 2022.


Michael Keeling. *11 zasad projektowania architektury oprogramowania*. PWN, Warszawa, 2021.


J.H. Saltzer i M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975. DOI: 10.1109/PROC.1975.9939.


Izabela Matos. Security in Software Architecture: understand the importance. URL: https://blog.convisoappsec.com/en/security-in-software-architecture-understand-the-importance/ (termin wizyty 12.11.2024).

Źródła II

Jeff Crume. Cybersecurity Architecture: Five Principles to Follow (and One to Avoid). URL: https://www.youtube.com/watch?v=jq_LZ1RFPfU&list=PLOspHqNVtKADkWLFt9OcziQF7EatuANSY (termin wizyty 13. 11. 2024).

Pytania

?

Koniec

Dziękuję Państwu za uwagę!