**Polsko-Japońska Wyższa Szkoła Technik Komputerowych**

# Uwierzytelnienie i autoryzacja

Kazimierz Subieta, Edgar Głowacki
edgar.glowacki@pjwstk.edu.pl



## Co to jest bezpieczeństwo informacji? (1)

Zgodnie z U.S. National Information Systems Security Glossary:
**Bezpieczeństwo systemów informacyjnych (INFOSEC)** to ochrona przed nieuprawnionym dostępem do informacji lub jej modyfikacją.

Ochrona powinna obejmować:

- przechowywanie, przetwarzanie, trasmisję;
- zabezpieczenie przed odmową usługi (DoS – *denial of services*) wobec użytkowników upoważnionych (*authorized*) lub dostarczeniem usługi użytkownikom nieupoważnionym
- środki umożliwiające wykrycie, dokumentację oraz przeciwdziałanie zagrożeniom

Systemy informacyjne to pojęcie daleko szersze niż systemy komputerowe – informacja nie musi być przechowywana w formie elektronicznej.

## Co to jest bezpieczeństwo informacji? (2)

Ogólnie przyjmuje się, że na bezpieczeństwo informacji składają się 3 elementy – tzw. ***CIA triad*** (od angielskich odpowiedników tych pojęć):

- **poufność** (***confidentiality***) – informacja jest dostępna jedynie dla podmiotów do tego upoważnionych;
- **spójność/integralność** (***integrity***) – wszelkie nieuprawnione modyfikacje informacji są niedozwolone;
- **dostępność** (***availability***) – do informacji można uzyskać dostęp w każdych okolicznościach, które są dopuszczone przez politykę bezpieczeństwa informacji.

Do tej trójki z czasem dołączono także: **możliwość rozliczania/rozliczalność** (***accountability***), czyli ustalenia odpowiedzialnych za wykonane operacje.

3

## Co to jest bezpieczne oprogramowanie? (1)

**Powszechnie uważa się, że bezpieczne oprogramowanie powinno zapewnić:**

- **uwierzytelnienie** (***authentication***) – proces polegający na weryfikacji, czy strona, od której pochodzi żądanie usługi jest rzeczywiście tym, za kogo się podaje;
- **zezwolenie na dostęp/autoryzacja** (***access authorization***) – proces weryfikacji, czy strona żądająca usługi jest rzeczywiście do niej upoważniona;

Ani **uwierzytelnienie**, ani **autoryzacja** nie były objęte ogólną definicją bezpieczeństwa systemów informacyjnych.

Uwierzytelnienie i autoryzacja **pełnią bowiem służebną rolę** w stosunku do wymienionych cech bezpiecznego systemu informacyjnego.

- **poufność** (***confidentiality***) – tak, jak w definicji ogólnej – zapewnienie, by informacja była dostępna jedynie dla podmiotów do tego upoważnionych;
- **spójność** (***integrity***) – podobnie jak w definicji ogólnej – zapewnienie, że informacja nie zostanie zmodyfikowana w sposób nieuprawniony.
  Spójność czasami błędnie utożsamia się z **aktualnością** (***accuracy***) – zgodnością z bieżącym stanem rzeczywistym.

4

## Co to jest bezpieczne oprogramowanie? (2)

- **niezaprzeczalność** (***non-repudiation***) – umożliwia weryfikację, że strony biorące udział w wymianie informacji rzeczywiście brały w niej udział. Innymi słowy nadawca i odbiora informacji nie mogą się wyprzeć, że uczestniczyli w wymianie informacji w rolach, w których rzeczywiście występowali

Z kolei bez **niezaprzeczalności** trudno byłoby mówić o rozliczalności i częściowo spójności.

- **dostępność** (***availability***) – stosunek czasu, w jakim system jest w stanie obsłużyć żądanie klienta w danym przedziale czasu do całkowitej długości ściśle określonego przedziału czasu.
  Dostępność jest jednym z elementów najczęściej objętych tzw. ***service level agreements*** (SLA).
  Obecnie za najwyższy (i najdroższy) stopień dostępności jest ustalany na: 0,99999 (tzw. pięć dziewiątek).
  Na podstawie trywialnego wyliczenia można przekonać się, że tego rodzaju kontrakt ustala maksymalny dopuszczalny czas w jakim usługa nie może być na **nieco ponad 5 minut w ciągu całego roku!**

5

## Co to jest bezpieczne oprogramowanie? (3)

Poza wymienionymi cechami od bezpiecznego oprogramowania wymaga się czasami również zapewnienia **prywatności** (***privacy***), czyli możliwości sprawowania przez podmiot kontroli nad informacją dotyczącą jego samego.

‘*Privacy is the power to selectively reveal oneself to the world.*’, Eric Hughes

**Bezpieczeństwo jest typowym wymaganiem niefunkcjonalnym**, które nie dodaje wartości biznesowej do produktu.

Dla przedsiębiorstwa bezpieczeństwo oprogramowania jest **drugorzędne w stosunku do wsparcia udzielonego przez to oprogramowanie wobec kluczowych procesów biznesowych**.
Co wcale nie oznacza, że produkt zostanie zaakceptowany przez klienta, jeśli nie będzie zapewniał wymaganego (wystarczającego) poziomu bezpieczeństwa.

6

## Jaki jest w ogóle sens i cel tego TBO? (1)

**Pytanie polityczne** ☺
Wśród użytkowników panuje powszechne przekonanie, że bezpieczeństwo informacji skutecznie utrudnia organizacji wypełnianie jej misji – czyli celu, dla którego powstała.

Bezpieczeństwo narzuca użytkownikom **źle dobrane, zbędne i uciążliwe procedury postępowania**.

*„Powodem takiego stanu rzeczy jest fakt, że wielu projektantów myśli o systemach zabezpieczeń **podobnie jak kucharz o przepisie kulinarnym** – cała sztuka polega na odpowiednim doborze składników (technologii) i ich integracji”, Bruce Schneier*

**Co gorsza uważa się, że bezpieczeństwo można dołożyć na samym końcu (!)** – na etapie wdrożenia oprogramowania

Prawie na pewno zarówno stworzony wg powyższych reguł system bezpieczeństwa informacji, jak i wchodzące w jego skład oprogramowanie nie będzie posiadało **dwóch zasadniczych cech decydujących o sukcesie i skuteczności** tego rodzaju systemów:

- możliwie **„bezszwowego”** (***seamless***) wkomponowywania się w zasadniczą działalność danej organizacji

7

## Jaki jest w ogóle sens i cel tego TBO? (2)

- uodparniania system informacyjny na zagrożenia, na które rzeczywiście może on być narażony.
  Przykładowo w niektórych modelach zagrożeń daleko istotniejsza jest analiza ruchu – tj. fakt, że określone strony (osoby – *participants*) komunikują się ze sobą – niż sama treść komunikacji.

Najpoważniejsze problemy związane z bezpieczeństwem **nie wynikają z błędów popełnionych na etapie wdrożenia a jego wytwarzania**.

Z reguły **im wcześniej taki błąd zostanie popełniony, tym jest poważniejszy i trudniejszy do usunięcia**.

**Inżynierii bezpieczeństwa** (***security engineering***) najbliżej jest do **inżynierii oprogramowania.**
Tak jak w przypadku inżynierii oprogramowania należy bowiem zacząć od **określenia wymagań** – czyli **zagrożeń na jakie system rzeczywiście może być narażony**.

8

# Uwierzytelnienie (1)

**Uwierzytelnienie** – proces, podczas którego jest ustalana tożsamość podmiotu.
U. zazwyczaj wiąże się z jakąś formą dostarczenia dowodu potwierdzającego tożsamość.
U. musi być wykonywane w sposób bezpieczny, tj. **uniemożliwiający możliwość podszycia** (np. zabezpieczenia dokumentów tożsamości)

Odpowiedź na pytanie: „Kim jesteś i jak możesz to udowodnić?" pozwala określić relację między stronami.

**3 rodzaje dowodów potwierdzających tożsamość (*credentials*) – w ogólności jak również w odniesieniu do rozwiązań informatycznych:**

- **Co użytkownik wie** – w przypadku systemów informatycznych najpowszechniej stosowany i jednocześnie najsłabszy mechanizm zabezpieczenia.
  O atakach słownikowych na hasła słyszał (chyba) każdy.
  Niestety prawo Moore'a (dot. wydajności przetwarzania procesorów) skutecznie zmniejsza odporność tej metody uwierzytelnienia na *brute force attack*.
  Trudno jest wymagać od użytkowników zapamiętania 32-znakowy losowy ciąg w zapisie szestnastkowym – specjalnie zresztą nie ma takiej potrzeby.

9

# Uwierzytelnienie (2)

**3 rodzaje dowodów potwierdzających tożsamość (c.d.):**

- **Co użytkownik wie** (c.d.)
  Można również stosować różnego rodzaju mnemotechniki – czyli wykorzystywać całe frazy – jakoś zmodyfikowane.
  Warto również zauważyć **różnicę między interfejsem manualnym** (zapezpieczonym 4-cyfrowym PIN-em) a **interfejsem elektronicznym**.
- **Czym użytkownik jest** – rozpoznawanie po cechach osobniczych to najstarszy sposób uwierzytelnienia (nie dotyczy wyłącznie ludzi).
  Uznawany za najbezpieczniejszy – ale wcale niekoniecznie jest to słuszne, cecha osobnicza raz skradziona jest skradziona **dożywotnio!**
  A wcale nie tak trudno jest ją skraść (np. odcisk palca).
  Cena czytników biometrycznych jest uzależniona od ich skuteczności – te naprawdę niezawodne są **naprawdę drogie**.
  Dostępne rodzaje czytników biometrycznych:
  - czytniki linii papilarnych (*fingerprint readers*),
  - czytniki wzoru tęczówki oka (*retinal scans*),
  - czytniki pisma odręcznego,
  - analizator spektrogramu głosu (*voiceprint*)

10

## Uwierzytelnienie (3)

**3 rodzaje dowodów potwierdzających tożsamość (c.d.):**

- **Co użytkownik ma** – czyli tzw. żetony dostępowe (*tokens*).
  Również bardzo stara forma uwierzytelnienia – pieczęć suwerena upoważniała do działania w jego imieniu.
  Przykłady żetonów (czasami zabezpieczonych hasłem – PIN-em):
  - karty magnetyczne – płatnicze, pełniące rolę kluczy do pokoi hotelowych;
  - karty zbliżeniowe;
  - karty mikroprocesorowe (np. z kluczem prywatnym RSA lub DSA);
  - mały kalkulatorek z klawiaturą numeryczną i wyświetlaczem pozwalający na wyliczenie kodu dostępowego w oparciu o *challenge* wyświetlony na ekranie komputera – rozwiązanie niegdyś wykorzystywane przez PeKaO S.A.
  - urządzenie posiadające jedynie wyświetlacz ciekłokrystaliczny, na którym w ściśle określonych odstępach czasu zmienia się wyświetlany kod;
  - żetonem jest również kartka zawierająca hasło – wbrew pozorom może to być bardzo bezpieczny (i tani) sposób uwierzytelnienia – np. długie hasło dzieli się na 2 części, z której jedną część należy zapamiętać.

***Two-factor authentication*** – rozwiązanie hybrydowe łączące przynajmniej dwóch z wymienionych sposobów uwierzytelnienia.

11

## W następnej części wykładu ...

**Mechanizmy uwierzytelnienia i autoryzacji dostępne w Java™:**

- piaskownica (*sandbox*) Java™
- Java™ Authentication and Authorization Service (JAAS)

12

## Standardowa autoryzacja w Java™ – *Java sandbox* (1)

**Zabezpieczenie oparte o własności uruchamianego kodu**

Piaskownica (***sandbox***) – jeden z najbardziej znanych mechanizmów zabezpieczających.
***Sandbox*** pozwala w prosty sposób zabezpieczyć środowisko uruchomienia kodu poprzez narzucenie mu szczególnego zestawu praw (*permissions*).

Zestaw uprawnień uruchamianego kodu określa tzw. polityka bezpieczeństwa (***security policy***).
Z wyjątkiem szczególnych okoliczności nie można jej zmodyfikować z działającego programu – co wydaje się dość oczywiste – inaczej trudno byłoby mówić o *sandbox*.

W domyślnej piaskownicy nie istnieje API, które umożliwiałoby modyfikację polityki bezpieczeństwa.



## Standardowa autoryzacja w Java™ – *Java sandbox* (1)

**Implementacja *sandbox* składa się z 3 elementów:**

- ***Security Manager*** (***SM***) – dostarcza mechanizmów używanych w API Java™ sprawdzających, czy żądana operacja nie narusza ustalonej przez administratora polityki bezpieczeństwa.
- ***Access Controller*** (***AC***) – znajduje się poniżej – jest dostawcą usług dla ***SM***, który w trakcie weryfikacji praw wykonania danych operacji korzysta z funkcjonalności dostarczanej przez *Access Controller*.
- ***Class Loader*** (***CL***) odpowiedzialna za ładowanie do JRE strumieni danych (np.plików) i konwertowania ich na na definicje klas.
  *Class Loader*:
  - hermetyzuje informację o ładowanych klasach, współpracuje z JRE przy określaniu przestrzeni nazw.
    Dla każdego ***codebase*** (np. `file:///usr/lib/java`, czy `http://evilSite.com`, ...) tworzona jest osobna instancja ***Class Loader***.
    Dzięki takiemu mechanizmowi nie jest możliwe podmienianie klas pochodzących z innych *codebases*.
  - wywołuje *Security Manager* w celu sprawdzenia, czy kod żądający (załadowania) jakiejś klasy rzeczywiście ma do niej dostęp.

## Standardowa autoryzacja w Java™ – *Java sandbox* (2)

Funkcjonalność ***Security Manager*** i ***Access Controller*** jest zbliżona – w wielu miejscach ***SM*** stanowi *wrapper* dla ***AC***.
Powody istnienia obu mechanizmów są głównie natury historycznej.

Klasa **`SecurityManager`** udostępniająca fragment funkcjonalności ***SM*** była do tej pory głównym interfejsem dostępu do mechanizmów bezpieczeństwa i z tego powodu nie mógł on ulec zmianie.

***SM*** pełni ważną rolę w trakcie definiowania i wymuszania (*enforecment*) polityki bezpieczeństwa.

**Zasadniczy algorytm wykorzystywany przez API Java™ w momencie wykonywania potencjalnie niebezpiecznych operacji**:

1. Programista korzysta w swoim kodzie z operacji udostępnianej w API Java™
2. API Java™ zasięga informacji od ***SM***, czy zlecona operacja jest dozwolona zgodnie z obowiązującą polityką bezpieczeństwa
3. Jeśli operacja jest dozwolona API Java™ *realizuje żądanie programisty*

3a. Jeśli wykonanie operacji nie jest dopuszczone *SM* zgłasza do środowiska **`SecurityException`**

15

## Standardowa autoryzacja w Java™ – *Java sandbox* (3)

```
/** poniżej poglądowo - jak to wygląda w implementacji */
try {
   FileReader fr = new FileReader("nazwa-pliku");
   BufferedReader bf = new BufferedReader(fr);
   ...
} catch (IOException e) {
    System.err.println(e); System.exit(-1);
}
...
```

**`FileReader`** tworzy przy tej okazji wewnętrzny obiekt **`FileInputStream`**, co wiąże się z zapytaniem do ***SM***, czy operacja taka jest dopuszczona.

```
public FileInputStream(String n) throws FileNotFoundException {
   SecurityManager sm = System.getSecurityManager();
   if (sm != null) {
      sm.checkRead(n);
   }
   ...
} /** the scheme for verifying the code permissions */
```

16

## Standardowa autoryzacja w Java™ – *Java sandbox* (4)

Wykonanie potencjalnie niebezpiecznej operacji **każdorazowo wiąże się z zapytaniem o zgodę *SM*** – nawet jeśli przed chwilą wykonanie danej operacji zostało, bądź nie zostało dopuszczone przez ***SM***.

Taka (pozorna) nieefektywność wynika z bardzo uzasadnionych powodów:
- polityka bezpieczeństwa mogła się zmienić od czasu ostatniego wywołania
- decyzja o tym, czy wywołanie jest dopuszczone, czy też nie zależy od kontekstu

Domyślnie aplikacje Java™ nie mają zainstalowanego ***SM***, który jest uruchamiany dopiero w przypadku użycia opcji **`-Djava.security.manager`** JRE.
Z kolei **`appletviewer`** zainstalowany w przeglądarce internetowej uruchamia bardzo restrykcyjnie skonfigurowany ***SM***.

17

## Standardowa autoryzacja w Java™ – *Java sandbox* (5)

**5 elementów wchodzących w skład konfiguracji *sandbox*:**

- **prawa** (***permissions***) określających rodzaje akcji, które mogą być wykonane przez kod.
  Prawa są instancjami klas rozszerzających abstrakcyjną klasę **`Permission`**.
  *Permission* jest określane przez:
  - typ: **`AllPermission`**, **`FilePermission`**, **`RuntimePermission`**, ...)
  - nazwę – nazwa pliku dla **`FilePermission`**; niektóre typy praw nie wymagają podawania nazwy (**`RuntimePermission`**, **`AllPermission`**)
  - akcję: **`"read"`**, **`"write"`** dla **`FilePermission`**; **`"stopThread"`** dla **`RuntimePermission`**
- **źródła kodu** (***code sources***) (klasa **`CodeSource`**) informujące o miejscu, skąd będzie uruchamiany kod (***code base***) oraz przez kogo został on podpisany.
  ***Code base*** jest podana w postaci URI:
  `http://pjwstk.edu.pl/`; `file:/C:/files/jdk1.5/`;
  `file:///files/jdk1.5/` (jeżeli korzystamy z domyślnego napędu)
  `http://pjwstk.edu.pl/app/` - klasy w danym katalogu
  `http://sun.com/app.jar` - klasy wewnątrz java archive
  `http://sun.com/*` - klasy i pliki jar w danym katalogu (bez podkatalogów)
  `http://sun.com/-` - klasy i pliki jar w danym katalogu (również podkatalogi)

18

## Standardowa autoryzacja w Java™ – *Java sandbox* (6)

**5 elementów wchodzących w skład konfiguracji *sandbox* (c.d.):**

- **domeny ochrony** (***protection domains***) (klasa **`ProtectionDomain`**) – definicje związków między ***code sources*** a ***permissions*** – określają prawa dla konkretnych *code sources*
- **repozytoria kluczy** (***keystores***) – podpisywanie kodu jest jednym ze sposobów zapewnienienia mu większej swobody działania.
  Chcąc podpisać kod naszej aplikacji powinniśmy skorzystać z dostarczanego z JSDK narzędzia **`jarsigner`**
  Kod pochodzący z zaufanej organizacji (podpisany przez taką organizację) może liczyć na większe przywileje.
  Skuteczność tego rodzaju mechanizmu zależy od certyfikatów klucza publicznego, których bezpieczne przechowywanie wiąże się z dodatkowymi czynnościami administracyjnymi.
  ***Keystores*** służą do przechowywania certyfikatów publicznych X.509.
  Każdemu certyfikatowi towarzyszy **alias**, z którego korzysta się w trakcie definicji polityki bezpieczeństwa.

19

## Standardowa autoryzacja w Java™ – *Java sandbox* (7)

**5 elementów wchodzących w skład konfiguracji *sandbox* (c.d.):**

- **polityka bezpieczeństwa** (***security policy***) – konstruuje *sandbox*.
  Jest to lista domen ochrony.
  Domyślnie wykorzystywaną implementacją polityki bezpieczeństwa jest klasa **`PolicyFile`**, można to zmienić poprzez ustawienie właściwości **`policy.provider`**.
  Do edycji plików polityki bezpieczeństwa wspieranych przez **`PolicyFile`** służy narzędzie **`policytool`**.
  Pliki polityki bezpieczeństwa mają bardzo prostą składnię i można je bez przeszkód modyfikować za pomocą zwykłego edytora tekstu.

20

## Standardowa autoryzacja w Java™ – *Java sandbox* (8)

**Przykładowy plik polityki bezpieczeństwa:**

```
keystore "${user.home}${/}.keystore";

grant codeBase "http://pjwstk.edu.pl/" {
   permission java.io.FilePermission "/tmp", "read";
   permission java.lang.RuntimePermission "queuePrintJob";
};

grant signedBy "edek", codeBase "file:${java.home}/lib/ext/." {
   permission java.security.AllPermission;
};

grant signedBy "edek", codeBase "file:${java.class.path}/." {
   permission java.net.SocketPermission "*:1024-",
      "accept, connect, listen, resolve";
};

grant {
   permission java.util.Permission "java.version", "read";
};
```

21

## Java Authentication and Authorization Service (1)

Usługa uwierzytelniania i autoryzacji dostarczana przez Java™.
Szkielet (***framework***), w oparciu o który można tworzyć rozwiązania wymagające zezwolenia na wykonywanie przez użytkownika określonych operacji.

**Zastosowanie JAAS:**

- **uwierzytelnienie podmiotów** (***subjects***) – użytkowników/usług
- **autoryzacja podmiotów** – w celu ustalenia, czy mają uprawnienia (*permissions*) umożliwiające wykonanie zleconych przez nich akcji
- wszędzie tam, gdzie korzystamy z rozwiązań opartych o Java™:
  - samodzielne aplikacje,
  - applety,
  - komponenty EJB, czy
  - servlety

**Architektura JAAS:**

- nie jest wbudowana w standardowy *framework* bezpieczeństwa Java™;
- JAAS stanowi uzupełnienie dla standardowej architektury bezpieczeństwa dostępnej w Java™ (*sandbox*) określającą prawa dostępu wykonywanego kodu na podstawie jego cech – nie na podstawie informacji, kto uruchamia dany kod.

22

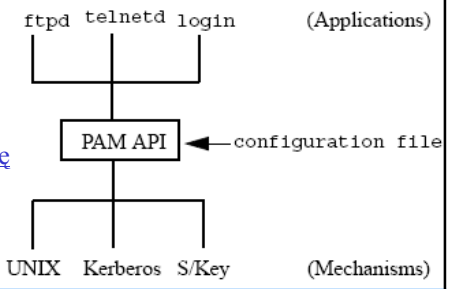
## Java Authentication and Authorization Service (2)

JAAS jest oparty na architekturze umożliwiającej **podłączanie modułów uwierzytelniających**, czyli tzw. PAM (***pluggable authentication modules***).

Pomysł PAM narodził się w roku 1996 w firmie Sun Microsystems i był efektem konieczności powstania mechanizmu umożliwiającego:

- elastyczną (dowolnie konfigurowalną) integrację różnych metod dostępu do systemu (*points of entry*) z różnymi metodami uwierzytelnienia – wówczas: (1) systemu haseł Unix, (2) NIS/NIS+, (3) Kerberos, itp.
- jednocześnie łączenia wielu metod uwierzytelnienia w trakcie dostępu do systemu – tzw. **„wrzucanie na stos”** modułów logowania

Podobnie jak w przypadku PAM (rysunek) JAAS umożliwia swobodne podłączanie i odłączanie modułów w oparciu o konfigurację – domyślnie plik konfiguracyjny.




23

## Uwierzytelnienie w JAAS (1)

**Scenariusz uwierzytelnienia z punktu widzenia developera:**

1. Tworzona jest instancja kontekstu logowania (`LoginContext`)
   Jako parametry wywołania należy przekazać:
   - nazwę wpisu, który w konfiguracji specyfikuje sposób uwierzytelnienia dla danej aplikacji,
   - instancję klasy reprezentującej **wywołanie zwrotne** (***callback***)
2. Wywołanie metody **`login()`** na rzecz stworzonego kontekstu.
   Metoda `login()` w oparciu o konfigurację dostarczoną przez administratora wywołuje odpowiednie moduły logowania (`LoginModule`)
   Efekt wywołania metody `login()` może być bardzo skomplikowany i zależy od konfiguracji logowania
3. Moduły logowania realizują czynności związane z uwierzytelnieniem, w czasie których mogą zarządać interakcji z podmiotem w celu dostarczenia dowodów tożsamości (***evidence***).
   Jedynym sposobem na przeprowadzenie interakcji jest **wywołanie zwrotne**.
   Pomyślne wywołanie metody login powoduje wypełnienie kontekstu obiektem reprezentującym podmiot (`Subject`) logowania.
   Podmiot zawiera zbiór charakteryzujących go cech dwóch rodzajów: ***principals***, ***credentials***.

24

## Uwierzytelnienie w JAAS (2)

***Principals*** reprezentowane jako obiekty klas implementujących interfejs **`Principal`** identyfikują podmiot w ramach środowiska wykonania aplikacji.
***Credentials*** – realizowane jako obiekty dowolnego typu – są wykorzystywane do dalszego uwierzytelnienia (hasła, certyfikaty klucza publicznego, bilety Kerberos...)

Implementacje *credentials* nie muszą zawierać właściwych danych związanych z dalszym uwierzytelnieniem, np. klucze prywatne przechowywane na karcie RSA. Istnieją 2 kategorie *credentials*:

- **publiczne** (certyfikaty klucza publicznego),
- **prywatne** (hasła, klucze wykorzystywane do szyfrowania lub podpisywania, bilety Kerberos)

***Credentials*** umożliwiają realizację **pojedynczego logowania** (***SSO*** – ***single sign-on***).
**SSO** – rodzaj uwierzytelnienia umożliwiający dostęp użytkownika do wielu zasobów po jednokrotnym logowaniu.

Wadą SSO jest tzw. ***single point of failure*** – naruszenie (*compromising*) systemu uwierzytelnienia powoduje dostęp do wszystkich zasobów chronionych przez SSO.

25

## Uwierzytelnienie w JAAS (3)

Konfiguracja środowiska zawiera wpisy określające zachowanie w trakcie uwierzytelnienia dla poszczególnych (fragmentów) aplikacji/modułów.
Specyfikacja zachowania polega na zdefiniowaniu kolejności wywołania oraz rodzajów zależności między modułami logowania.

Poniżej przedstawiono składnię pliku konfiguracyjnego wspierana przez **`ConfigFile`** – domyślną implementację **`Configuration`**.

```
...
<Application-or-Module> {
   <ModuleClass> <Flag> <ModuleOptions>;
   ...
};
...
```

Zmiana domyślnej implementacji (**`ConfigFile`**) odpowiedzialnej za konfigurację uwierzytelnienia JAAS jest możliwa poprzez ustawienie właściwości: **`login.configuration.provider`**.

26

## Uwierzytelnienie w JAAS (4)

**Zależności między modułami określają flagi w konfiguracji.**
Wyróżniamy 4 rodzaje flag:

- ***required*** – uwierzytelnienie przez moduł musi się zakończyć sukcesem
  *fail/succeed* → **`LoginContext`** kontynuuje przetwarzanie listy modułów
- ***requisite*** – uwierzytelnienie musi zakończyć się sukcesem
  *succeed* → **`LoginContext`** przechodzi do następnego modułu
  *fail* → *authentication failed* – nie przechodzi do następnego modułu
- ***sufficient*** – uwierzytelnienie nie musi zakończyć się powodzeniem.
  *succeed* → *authentication succeeded*,
  *fail* → przetwarzanie listy modułów jest kontynuowane
- ***optional*** – uwierzytelnienie nie musi zakończyć się sukcesem.
  *fail/succeed* → lista modułów jest dalej przetwarzana.

Cały proces uwierzytelnienia kończy się powodzeniem jeśli:
- wszystkie ***required*** i ***requisite*** zakończą się sukcesem,
- sukcesem zakończy się ***sufficient*** oraz wszystkie ***required*** i ***requisite*** przed nim,
- jeśli nie ma ***required*** i ***requisite*** – przynajmniej jeden ***sufficient*** lub ***optional*** zakończą się powodzeniem

27

## Uwierzytelnienie w JAAS (5)

Opcje dla modułu w konfiguracji modułów logowania.
W przypadku **`ConfigFile`** jest to oddzielona białymi znakami lista w postaci:

```
<klucz>="<wartość>"
```

przekazywana bezpośrednio do modułu logowania.

**Wołania zwrotne** (***callbacks***) są jedynym dostępnym mechanizmem umożliwiającym komunikację między modułem logowania a uwierzytelnianym podmiotem.

Jeśli **`LoginModule`** potrzebuje przeprowadzić interakcję z podmiotem, wypełnia odpowiednio tablicę zawierającą żądania wywołań zwrotnych (interfejs **`Callback`**), np. **`NameCallback`**, **`PasswordCallback`**.
Pamiętamy, że w trakcie tworzenia obiektu **`LoginContext`** podaliśmy instancję naszego **`CallbackHandler`**, który w metodzie **`handle()`** ustawi odpowiednie właściwości wołań zwrotnych: **`setName()`**, **`setPassword()`**

Mechanizmu wołań zwrotnych używa się zarówno w celu przekazania dowodów tożsamości (*evidence* w terminologii JAAS, czyli ogólnie *credentials*), jak również wysyłania komunikatów (np. o błędzie) do podmiotu przez **`LoginModule`**.

28

## Uwierzytelnienie w JAAS (6)

**Dwuetapowe uwierzytelnienie w JAAS:**

JAAS podobnie jak PAM wspiera mechanizm **„wrzucania na stos”** modułów logowania.

1. W celu zapewnienia, że „transakcja” uwierzytelniająca zakończy się powodzeniem lub sukcesem `LoginContext` wywołuje na każdym z `LoginModule` metodę `login()` ograniczającą się do realizacji uwierzytelnienia – **bez wypełnienia *principals* i *credentials***.
2. Dopiero **pomyślne zakończenie** całego procesu uwierzytelnienia (patrz. wcześniej) wiąże się z wywołaniem metody `commit()`, które wypełniają właściwości odpowiednimi wartościami.

W przypadku, gdy pierwsza, bądź druga faza zakończy się niepowodzeniem `LoginContext` zleca modułom logowania wyczyszczenie wpisów (***principals*** i ***credentials***) związanych z uwierzytelnieniem wywołując metodę `abort()`.

29

## Uwierzytelnienie w JAAS (7)

**Przykład pliku konfiguracyjnego dla uwierzytelnienia z użyciem framework JAAS:**

```
/* konfiguracja Sample koresponduje z przedstawioną w dalszej
 * części konfiguracją autoryzacji w oparciu o JAAS */
Sample {
   com.sun.security.auth.module.Krb5LoginModule required;
};

CountFiles {
   com.sun.security.auth.module.SolarisLoginModule required;
   pl.edu.pjwstk.PolJapLoginModule requisite;
   com.sun.security.auth.module.NTLoginModule sufficient;
   com.sun.security.auth.module.SolarisLoginModule required;
};

DBApplication {
   com.sun.security.auth.module.SolarisLoginModule optional;
   com.sun.security.auth.module.NTLoginModule sufficient;
};
```

30

# Autoryzacja w JAAS (1)

Po pomyślnym zakończeniu fazy uwierzytelnienia JAAS udostępnia mechanizmy wymuszające kontrolę dostępu w oparciu o ***principals*** związanych z przedmiotem uwierzytelnienia.

**Kontrola dostępu oparta o cechy podmiotu wywołującego** (***principal-based access control***)
Autoryzacja JAAS jest oparta na dokładnie tym samym (dowolnie rozszerzalnym) zestawie praw i przebiega analogicznie jak w przypadku *sandbox*.

Zasadnicza różnica polega na kryterium podejmowania decyzji – w przypadku piaskownicy podstawą na zezwolenie wykonania operacji były własności kodu, w JAAS decydują **własności podmiotu wywołującego** (*principals*).

**Przebieg autoryzacji z wykorzystaniem framework JAAS:**

0. Jak pamiętamy, po pomyślnie zakończonym uwierzytelnieniu posiadamy `LoginContext` z wypełnionym obiektem reprezentującym podmiot wywołujący operację (`Subject`).
   `Subject` zawiera wewnątrz właściwości:
   - identyfikujące go w ramach JAAS (*principals*), oraz
   - umożliwiające uwierzytelnienie wobec innych *authorities*, lub wykonywanie innych operacji (*credentials*)



# Autoryzacja w JAAS (2)

**Przebieg autoryzacji z wykorzystaniem framework JAAS (c.d.):**

1. Program tworzy instancję klasy implementującej **`PriviledgedAction`**, która reprezentuje akcję, którą chce wykonać w imieniu użytkownika.
   Interfejs **`PriviledgedAction`** wymaga implementacji metody **`run()`**, która jest odpowiedzialna za realizację zadania zleconego przez **`Subject`**.
2. Program wykonuje metodę: **`doAs()`** lub **`doAsPriviledged()`** przekazując **`Subject`** oraz stworzoną instancję **`PriviledgedAction`**
3. Jeśli wywoływana operacja jest potencjalnie niebezpieczna, w pierwszej kolejności weryfikowane są prawa dostępu kodu (***code-based access control***, czyli *sandbox*) – o zgodę na wykonanie jakiejś akcji jest pytany jest *Security Manager*.
   *SM* podejmuje decyzję o dopuszczeniu lub odrzuceniu żądania kodu w oparciu o informację przekazaną przez *Access Controller*.
4. Jeśli *SM* nie zgłosi zastrzeżeń do wykonania kodu kontrolę przeprowadza szkielet JAAS, który sprawdza w swojej polityce bezpieczeństwa, czy z kolei kontekst użytkownika pozwala na wykonanie żądanej operacji.
   *Framework* JAAS również wykorzystuje do tego celu *Access Controller*.

# Autoryzacja w JAAS (3)

**Przebieg autoryzacji z wykorzystaniem framework JAAS (c.d.):**

5. Jeśli polityka bezpieczeństwa umożliwia uruchomienie wywoływana jest metoda **`run()`** na obiekcie **`PriviledgedAction`**

W trakcie korzystania z autoryzacji JAAS należy pamiętać o fakcie istnienia dwóch różnych rodzajów polityki bezpieczeństwa:

- standardowej, która definiuje działanie ***Security Manager***
- rozszerzającej tą pierwszą – polityki bezpieczeństwa JAAS

**Przykładowa realizacja akcji uprzywilejowanej:**

```
import java.security.PrivilegedAction;

public class SampleAction implements PrivilegedAction {
   public Object run() {
      System.out.println(
         "java.home value: " + System.getProperty("java.home")
       );
   }
}
```

33

# Autoryzacja w JAAS (4)

**Specyficzne rodzaje praw dla kodu konfiguracyjnego JAAS:**

**Kod konfiguracyjny JAAS** – kod odpowiedzialny za przygotowanie środowiska JAAS na potrzeby realizacji zlecenia pochodzącego od danego **`Subject`**.

Kod konfiguracyjny JAAS powinien mieć przydzielone specyficzne prawa w zależności od wykonywanych zadań

1. Kod konfiguracyjny odpowiedzialny za **stworzenie `LoginContext`**, oraz **wywołanie uprzywilejowanej akcji w imieniu użytkownika** powinien mieć przyznane odpowiednio następujące prawa (oczywiście poza innymi prawami w zależności od wykonywanych innych operacji):

```
javax.security.auth.AuthPermission "createLoginContext";
javax.security.auth.AuthPermission "doAs";
javax.security.auth.AuthPermission "doAsPriviledged";
```

2. Z kolei implementacja **`LoginModule`** powinna mieć z kolei – przynajmniej – następujące prawa:

```
javax.security.auth.AuthPermission "modifyPrincipals";
```

34

## Autoryzacja w JAAS (5)

Należy w tym miejscu zauważyć, że.
Najprawdopodobniej nie jest naszą intencją by kod wykonywany w imieniu użytkownika mógł dowolnie zestawiać połączenia sieciowe, lub uruchamiać **zazwyczaj kod konfiguracyjny JAAS musi mieć większe prawa niż kod uruchamiany w imieniu użytkownika** dowolne zewnętrzne aplikacje.

*Codebase* kodu konfiguracyjnego JAAS (np. odpowiedzialnego za uwierzytelnienie podmiotu) powinien być różny od *codebase* kodu uruchamianego w imieniu podmiotu.
**Z tego względu aplikacja współpracująca z JAAS powinna być podzielona na co najmniej 2 *java archives*.**



## Autoryzacja w JAAS (6)

**Przykład pliku konfiguracji autoryzacji w JAAS korespondujący z zaprezentowaną wcześniej konfiguracją uwierzytelnienia:**

```
/* protection domain zezwalająca na stworzenie kontekstu
 * logowania dla wpisu "Sample" w konfiguracji
 * uwierzytelnienia */
grant codebase "file:./SampleAuthentication.jar" signed {
   permission javax.security.auth.AuthPermission
      "createLoginContext.Sample";
};

/* protection domain zezwalająca na wykonanie akcji w imieniu
 * użytkownika: odczytu właściwości "java.home", odczytu pliku
 * "foo.txt" */
grant codebase "file:./SampleAction.jar",
   Principal javax.security.auth.kerberos.KerberosPrincipal
      "user@realm"
{
   permission java.util.PropertyPermission "java.home", "read";
   permission java.io.FilePermission "foo.txt", "read";
};
```

## W następnej części wykładu ...

**Mechanizmy uwierzytelnienia i autoryzacji dostępne w Microsoft .NET**

37

## Podstawy .NET Framework – *assembly*

***Assembly*** – podstawowa jednostka, z której zbudowane są aplikacje pod .NET Framework.

1. *Assembly* jest kolekcją typów danych i różnego rodzaju innych zasobów (pliki HTML, XML, pliki graficzne, ***assembly resource files***, itp.), które razem dostarczają pewną funkcjonalność.
2. *Assembly* dostarcza CLR wszystkich informacji nt. implementacji zawartych w niej typów, czyli: metadane + kod w MSIL. Z punktu widzenia CLR każdy typ występuje w kontekście *assembly*.
3. *Assembly* określa zakres odwołania (***scoping***). Manifest *assembly* zawiera metadane używane do rozwiązania żądań typów i zasobów. Manifest określa:
   - elementy udostępniane przez *assembly* na zewnątrz,
   - inne *assemblies*, od których dane *assembly* zależy.
4. *Assembly* określa granice bezpieczeństwa – w polityce bezpieczeństwa prawa dostępu są przydzielane na poziomie *assembly*.
5. Z perspektywy CLR *assembly* jest najmniejszą wersjonowaną jednostką – zawartość *assembly* jest wersjonowana jako całość – manifest assembly zawiera wersje zależnych *assemblies*.
6. *Assembly* jest podstawową jednostką wdrożenia aplikacji.

38

## Podstawy .NET Framework – *application domain* (1)

Domena aplikacji (**`AppDomain`**) – abstrakcyjne pojęcie oznaczające **izolowane środowisko, w którym uruchamiane są *assemblies***.

**`AppDomain`** jest mechanizmem ortogonalnym do wszystkich znanych do tej pory sposobów izolacji wykonywanego kodu.
**`AppDomain`** **nie jest związana granicami z**:

- komputerem (*host*), na którym uruchomiono aplikację;
- procesem, w którym wykonywany jest kod aplikacji;
- wątkiem procesu systemu operacyjnego w ramach, którego jest wykonywany kod aplikacji.

**W danym momencie wątek procesu jest wykonywany w jednej `AppDomain`**, ale w ogólności może być wykonywany w wielu **`AppDomains`**.

W ramach jednego procesu (jedno- lub wielowątkowego) może istnieć wiele **`AppDomain`**.
Stopień izolacji w ramach **`AppDomain`** jest **tożsamy z izolacją między procesami** w tradycyjnych systemach.
Ale nie istnieje potrzeba komunikacji międzyprocesowej, jeśli **`AppDomains`** zostały utworzone w ramach jednego procesu.

39

## Podstawy .NET Framework – *application domain* (2)

Błędy powstałe w jednej **`AppDomain`** **nie wpływają na działanie kodu uruchomionego w innych `AppDomains`**.
Z pamięci operacyjnej może być usunięta jedynie cała **`AppDomain`** – nie ma możliwości usunięcia pojedynczych *assemblies* zawartych w **`AppDomain`**.

Nie ma możliwości bezpośrednich wołań między obiektami znajdującymi się w różnych **`AppDomains`**.
**Jedynym sposobem komunikacji między obiektami z różnych `AppDomains` jest wołanie przez .NET Remoting.**

.NET Remoting stanowi abstrakcyjne podejście do komunikacji międzyprocesowej (między **`AppDomains`**) oddzielający pojęcie **obiektu udostępniającego usługi** od:

- rodzaju aplikacji – zarówno klient jak i serwer mogą być aplikacjami dowolnego typu;
- wykorzystywanych mechanizmów w komunikacji między klientem i serwerem.

40

## Podstawy .NET Framework – *application domain* (3)

**Komunikacja między obiektami znajdującymi się w różnych `AppDomains`** może odbywać się na dwa różne sposoby:

- poprzez przekazywanie wartości (***marshall by value***) – między **`AppDomains`** przesyłane są kopie obiektów, na których rzecz wywoływana jest metoda.
- poprzez przekazywanie referencji (***marshall by reference***) – w trakcie pierwszego dostępu do obiektu zdalnego od strony obiektu zdalnego jest przesyłany obiekt ***proxy*** reprezentujący obiekt zdalny po stronie klienta. Wywołania lokalne na obiekcie proxy są przekazywane (*marshalled*) do obiektu zdalnego.

Zdalne wywołanie metody przez .NET Remoting wymaga, by typ obiektu, na rzecz którego wywoływana jest metoda rozszerzał **`System.MarshalByRefObject`**, co umożliwia przesłanie referencji do obiektu kanałem transportu.

Jeśli typ obiektu nie rozszerza **`System.MarshalByRefObject`** .NET Framework domyślnie będzie próbował przekazać go przez wartość – wymagana jest w tym celu szeregowalność (***serializability***) obiektu.
Przy przekazaniu przez wartość metoda obiektu zostanie wywołana lokalnie – w **`AppDomain`**, z której wywoływano metodę obiektu.

41

## Podstawy .NET Framework – *application domain* (4)

**Przykład implementacji wywołania zdalnego przez granice `AppDomain`.**

```
/// Class Greetings.Greet which allows for remote calls of its
/// objects across AppDomains boundaries.
/// Greetings.Greet should be placed in assembly:
/// Greeetings.dll (see further) which should be available in
/// bothAppDomains

namespace Greetings {

   public class Greet : MarshalByRefObj {
      public Greet () {}

      public Say () {
         Console.WriteLine(
            "Hello World in the application domain: "
            + Thread.GetDomain().FriendlyName );
      }
   }
}
```

42

## Podstawy .NET Framework – *application domain* (5)

```
/// Class invoking a remote method Say() of Greetings.Greet.
/// Class InvokeRemoteMethod should compose a separate assembly
/// e.g. InvokeRemoteMethod.exe.
/// GreetingsAssembly.dll must be accessible for
/// InvokeRemoteMethod.exe as well.
public class InvokeRemoteMethod {
   public static void Main (string[] args) {
      AppDomainSetup setup = new AppDomainSetup();
      setup.ApplicationBase = "file:///"
         + Environment.CurrentDirectory;
      AppDomain domain = AppDomain.CreateDomain(
         "GreetingsDomain", null, setup);
      Object o = domain.CreateInstanceFromAndUnwrap(
         setup.ApplicationBase + "/" + "Greetings.dll",
         "Greetings.Greet" );
      ((Greetings.Greet)o).Say(); /// Hello World in the
      /// application domain ...
   }
}
```



## Podstawy .NET Framework – *runtime host* (1)

Zasadniczo .NET Framework składa się z dwóch głównych komponentów:

- **Common Language Runtime (CLR)** – rodzaj agenta zarządzającego kodem w trakcie wykonania dostarczającego podstawowe usługi: (1) zarządzanie pamięcią, (2) współbieżność (tworzenie i zarządzanie wątkami utworzonymi w ramach procesu), (3) oraz zdalne wywołania (*remoting*).
  Kod wykorzystujący własności CLR jest tzw. kodem zarządzanym (***managed***).
- **biblioteki klas** (***class library***) – spójny zestaw typów, które mogą być wykorzystywane do tworzenia własnych rozwiązań

***Runtime Host*** – składający się kodu zarządzanego i niezarządzanego (***unmanaged***) komponent odpowiedzialny za ładowanie **CLR** do procesu, w którym sam został uruchomiony.
Zadaniem *Runtime Host* jest przygotowanie środowiska, które pozwala wykorzystywać hostowanej aplikacji .NET funkcjonalność dostarczaną przez kod zarządzany jak i niezarządzany.

## Podstawy .NET Framework – *runtime host* (2)

**Scenariusz przygotowywania środowiska wykonania aplikacji przez *Runtime Host*:**

1. *RH* ładuje CLR do procesu, w którym został uruchomiony.
   W momencie inicjalizacji CLR tworzona jest tzw. domyślna **`AppDomain`**
2. *RH* dokonuje własnego przeniesienia (***transition***) z kodu niezarządzanego do kodu zarządzanego.
   Hostujący kod zarządzany (***managed hosting code***) *RH* jest umieszczany w domyślnej **`AppDomain`**.
   Zadaniem kodu hostującego jest interakcja z **CLR**.
   Dopiero po przeniesieniu *RH* do kodu zarządzanego jest możliwe tworzenie osobnych **`AppDomain`** dla kodu użytkownika.
   Przeniesienie *RH* do kodu zarządzanego wiąże się ze zwiększeniem wydajności – eliminowana jest konieczność przełączania między kodem zarządzanym i niezarządzanym przy komunikacji między *RH* i kodem użytkownika
3. *RH* tworzy **`AppDomains`** dla kodu użytkownika (aplikacji).
   Istnieje co prawda możliwość uruchomienia kodu użytkownika w domyślnej **`AppDomain`**, ale w ten sposób nie byłoby możliwości usunięcia go niezależnie od procesu hostującego – kod użytkownika byłby związany z *RH*.
4. *RH* ładuje *assemblies* aplikacji do odpowiednich **`AppDomains`**

45

## Podstawy .NET Framework – *runtime host* (3)

**.NET Framework jest dostarczany z 3 typami *Runtime Host*:**

- ASP.NET (IIS), które ładuje CLR do procesu, który ma obsłużyć żądanie webowe.
  Tworzy **`AppDomain`** dla każdej aplikacji uruchomionej na IIS.
- MSIE, dzięki któremu jest możliwe jest umieszczenie komponentów zarządzanych (podobnych do ActiveX) lub formularzy Windows w dokumentach HTML.
  Ten rodzaj *RH* umożliwia wykorzystanie cech dostępnych tylko w przypadku kodu zarządzanego: piaskownica (*semi-trusted execution*), czy izolowane repozytorium plików (*isolated file storage*)
- .NET EXE – *RH* dla programów uruchamianych z powłoki.

**Przykład izolacji przy użyciu AppDomains zapewnianej przez MSIE:**
Kod kontrolek wykonywanych przez przeglądarke ściągnięty z witryny internetowej jest izolowany od kodu ściągniętego z innych witryn.
Dla każdej witryny MSIE tworzy osobną **`AppDomain`**.

46

## Mechanizmy autoryzacji w .NET Framework

System zabezpieczeń dostępny w .NET Framework dostarcza mechanizmy oraz narzędzia umożliwiające ochronę dostępnych zasobów przed ich niedozwolonym wykorzystaniem przez kod złośliwy (*malicious*) – zazwyczaj pochodzący z nieznanego źródła.

Wspomniana ochrona jest oparta o **szereg ograniczeń, które mogą być zdefiniowane zarówno przez twórcę oprogramowania, jak i administratora**.

Uprawnienia uruchamianiego kodu są zależne od:
- stopnia zaufania, jakim obdarzy go administrator systemu,
- ograniczeń zdefiniowanych przez programistę – dla rozpatrywanego kodu,
- stopnia zaufania programisty do kodu korzystającego z funkcjonalności rozpatrywanego komponentu.

System bezpieczeństwa w .NET Framework – podobnie jak jego odpowiednik w Java™ – **opiera swoje zaufanie do kodu na dwóch rodzajach własności**:
- cechach samego kodu – ***evidence-based authorization***;
- w imieniu kogo (jakiego użytkownika, jakiej roli) kod miałby być uruchomiony – ***principal-based authorization***.

47

## Code-Access Security (1)

**Code-Access Security** jest mechanizmem przyznającym dostęp do zasobów w oparciu o tożsamość (***evidence***) kodu.
System zabezpieczeń .NET Framework musi zidentyfikować kod oraz miejsce jego pochodzenia (*origin*) zanim podejmie decyzję o jego wykonaniu.

***Evidence*** jest zestawem informacji pozwalającym na identyfikację oraz ustalenie miejsca pochodzenia kodu.

**Rodzaje *evidence* umożliwiające zidentyfikować kod:**
- Strong Name (unikalny klucz publiczny, krótka nazwa, oraz wersja assembly)
- skrót (*hash*)

**Rodzaje *evidence* umożliwiające ustalić miejsce pochodzenia kodu:**
- wydawca (*publisher*) assembly – może być zweryfikowany przez **Microsoft Authenticode** signature;
- strefa (*zone*) pochodzenia assembly: (1) lokalny komputer, (2) intranet, (3) internet;
- źródło pochodzenia: URI, ścieżka UNC (Universal Naming Convention), katalog na lokalnym komputerze.

Twórca oprogramowania może dodatkowo zdefiniować własne rodzaje *evidence*.

48

## Code-Access Security (2)

W czasie ładowania *assembly* **CLR** wykorzystuje hostujące je oprogramowanie w celu przedstawienia *evidence* towarzyszącego danemu *assembly* systemu zabezpieczeń .NET Framework.

System zabezpieczeń powinien przyznawać uprawnienia w oparciu o stopień wiarygodności (*strength*) zabezpieczenia.

Im mocniejsze *evidence*, tym mniejsze prawdopodobieństwo jego podrobienia.
Słabym *evidence* są URI, czy strefa pochodzenia (*zone*).
Szczególnie mocnym *evidence* jest tzw. **Strong Name**, ale oczywiście zależy to od stopnia zabezpieczenia klucza prywatnego wydawcy.
**Rozdzielenie procesu kompilacji assembly od podpisywania przed wydaniem (*release*) może skutecznie zwiększyć bezpieczeństwo *evidence*** poprzez ograniczenie liczby osób posiadających dostęp do klucza.

```
/// Below "attribute" allows to separate the compilation of the
/// assembly from signing before release.
/// The private code is accessible to selected personnel
[assembly: AssemblyDelaySign(true)]
```

49

## Code-Access Security (3)

***Code-access security policy*** – polityka bezpieczeństwa przyznająca uprawnienia *assembly* w oparciu o towarzyszące mu *evidence*.

Przyznawanie zestawu uprawnień w oparciu o evidence odbywa się jednokrotnie – na etapie ładowania assembly.
Zmiana polityki bezpieczeństwa, kiedy assembly jest już załadowane wymaga jego usunięcia i ponownego załadowania – pośrednio przez AppDomain.

**Rodzaje uprawnień (*permissions*) dostępnych w .NET Framework:**

- **`DirectoryServicePermission`** – dostęp do usług katalogowych;
- **`DnsPermission`** – możliwość korzystania z usługi nazwowej;
- **`EnvironmentPermission`** – dostęp do zmiennych środowiskowych;
- **`EventLogPermission`** – dostęp do systemowego rejestru zdarzeń;
- **`FileDialogPermission`** – okienka dialogowe do obługi plików (**Otwórz**, **Zapisz**, ...);
- **`IsolatedStoragePermission`** – możliwość korzystania z izolowanego składu;
- **`MessageQueuePermission`** – możliwość korzystania z usług kolejkowych Windows;

50

## Code-Access Security (4)

**Rodzaje uprawnień (*permissions*) dostępnych w .NET Framework (c.d.):**

- **`FileIOPermission`** – dostęp do plików i katalogów w systemie plików;
- **`OleDdPermission`** – dostęp do baz danych przez OLE DB;
- **`PerformanceCounterPermission`** – dostęp do liczników wydajności;
- **`PrintingPermission`** – dostęp do drukarek;
- **`ReflectionPermission`** – możliwość uzyskania informacji o typie w czasie wykonania;
- **`RegistryPermission`** – dostęp do rejestru systemowego Windows;
- **`SercurityPermission`** – możliwość wykonywania kodu, zagwaratnowania (assert) uprawnień, wołania kodu niezarządzanego, pomijania weryfikacji, ...;
- **`ServiceControllerPermission`** – uruchamianie i zatrzymywanie usług Windows;
- **`SocketPermission`** – zestawianie połączeń sieciowych przez gniazda;
- **`SqlClientPermission`** – dostęp do baz danych przez Microsoft SQL Server data access provider;
- **`UIPermission`** – dostęp, tworzenie elementów GUI;
- **`WebPermission`** – połączenia przez HTTP.

Powyższy zbiór może zostać rozszerzony przez twórców oprogramowania poprzez własną implementację klasy abstrakcyjnej **`CodeAccessPermission`**.

51

## Code-Access Security (5)

Wymienione rodzaje uprawnień są następne grupowane w zbiory uprawnień (*permission sets*).

**Domyślnie w .NET Framework zdefiniowano następujące zestawy uprawnień:**

- **Nothing** – brak jakichkolwiek uprawnień;
- **Execution** – możliwość uruchamiania, ale bez dostępu do chronionych zasobów;
- **Internet** – możliwość: uruchomienia, tworzenia posiadających szereg ograniczeń „bezpiecznych okienek”, plikowych okienek dialogowych, połączeń sieciowych z witryną pochodzenia assembly, korzystania z Isolated Storage o określonej pojemności;
- **LocalIntranet** – możliwość: uruchomienia, tworzenia interfejsu użytkownika oraz Isolated Storage bez ograniczeń, korzystania z DNS, odczytywania wybranych zmiennych środowiskowych, tworzenie połączeń sieciowych z witryną pochodzenia, czytanie i zapis do plików w tym samym katalogu co assembly;
- **Everything** – wszystkie wbudowane uprawnienia poza możliwością pominięcia weryfikacji;
- **FullTrust** – pełen dostęp do wszystkich zasobów.

52

## Code-Access Security (6)

Na podstawie ściśle określonych kryteriów przynależności *assemblies* mogą być łączone w ***code groups***, którym przyznaje się konkretne prawa dostępu.
**O przynależności do *code group* decyduje *evidence***.

**Rodzaje przynależności dostępne w .NET Framework:**

- ***application directory*** – wspólny katalog instalacji aplikacji;
- ***cryptographic hash*** – określony skrót MD5 lub SHA1;
- ***strong name*** – „prostsza wersja" **Authenticode** (bez PKI) – podpis złożony przez dostawcę obejmuje zawartość assembly, nazwę i wersję – gwarantuje unikalność;
- ***software publisher*** (**Authenticode**) – *assembly* podpisanemu kluczem prywatnym towarzyszy certyfikat X.509 wydawcy oprogramowania;
- ***URL*** – źródło pochodzenia assembly;
- ***web site*** – witryna, z której pochodzi assembly (nazwa DNS, lub adres IP);
- ***zone*** – strefa bezpieczeństwa, z której pochodzi assembly;

Polityka bezpieczeństwa może być ustalana różnie na poziomie całej organizacji niż na poziomie poszczególnych komputerów, czy użytkowników.
Na różnych poziomach można inaczej rozumieć zaufanego dostawcę oprogramowania.
Z tego powodu ***code-access security policy*** podzielono na **poziomy** (***policy levels***).

53

## Code-Access Security (7)

**W .NET Framework zdefiniowano cztery poziomy polityki bezpieczeństwa:**

- ***enterprise*** – ustalana przez administratorów domen;
- ***machine*** – ustalana przez administratorów konkretnego komputera;
- ***user*** – określana dla konkretnego konta użytkownika;
- ***application domain*** – określana przez *CLR host* – nie może być swobodnie ustalana z poziomu .NET Framework.

Polityka na poziomie *application domain* zazwyczaj zabrania bezpośredniego dostępu kodu wykonywanego w jednej `AppDomain` do kodu z innej `AppDomain`.
Przykładowo dostawca usług hostujący wiele aplikacji ASP.NET nie może dopuścić, by jedna aplikacja miała dostęp do innych hostowanych aplikacji.

*Code groups* są zorganizowane hierarchicznie na każdym poziomie bezpieczeństwa.
Na szczycie hierarchii jest ***all code***, które może mieć dowolną liczbę podgrup, które z kolei mogą mieć swoje podgrupy.
Przynależność do danej podgrupy jest sprawdzana tylko jeśli *assembly* spełnia kryterium nadgrupy.

54

## Code-Access Security (8)

**Przykładowe ustalanie przynależności do *code groups*:**
1. O przynależności do grupy A decyduje wydawca, np. Microsoft;
2. Do podgrupy A: A1 należy kod pochodzący z witryny *.microsoft.com.
Kod zakwalifikowany do A1 należy zawsze do A.

W momencie ładowania *assembly* jest przeprowadzana weryfikacja na okoliczność przynależności do wszystkich *code groups*, na wszystkich poziomach.

Assembly może zostać zakwalifikowane do:
- **wielu poziomów polityki bezpieczeństwa**;
- **wielu *code groups* na każdym poziomie polityki bezpieczeństwa**.

O ile nie zaznaczono, by na danym poziomie były uznawane tylko uprawnienia związane z daną grupą, .NET Framework domyślnie przyzna **sumę uprawnień wszystkich *code groups* na tym poziomie**.

55

## Code-Access Security (9)

Wynikowy zestaw uprawnień przyznany przez system zabezpieczeń .NET Framework **będzie iloczynem uprawnień dla wszystkich rozpatrywanych poziomów**.

W trakcie definiowania polityki bezpieczeństwa administrator może jawnie wymusić, by system zabezpieczeń .NET Framework nie brał w ogóle pod uwagę *code groups* zdefiniowanych na niższych poziomach.

Przykładowo:
- użytkownik przyznaje *assemblies* o danym zestawie evidence uprawnienia **FullTrust**,
- wcześniej administrator domeny określi prawa dostępu dla takiej *code group* na **Nothing**.

Wynikowe uprawnienia będą oczywiście **Nothing**, co daje gwarancję, że **polityki bezpieczeństwa zdefiniowane na różnych poziomach nie będą się nawzajem naruszać**.

56

## Code-Access Security (10)

**Przyznane uprawnienia muszą być następnie wyegzekwowane** dzięki czemu kod uzyska dostęp jedynie do zasobów określonych w polityce bezpieczeństwa.

W celu sprawdzenia, czy kod wywołujący *assembly* może korzystać bezpośrednio, bądź pośrednio z danego zasobu wykonywany jest **„spacer po stosie”** (***stack walk***).

***Stack walk*** odbywa się w dół stosu – począwszy od ostatniego rekordu aktywacji do najstarszego rekordu aktywacji (czyli wywołania `Main()`).
Przejście w dół stosu oznacza fizycznie przejście do wyższych adresów pamięci.
Dla każdego odwiedzonego rekordu aktywacji *stack walk* weryfikuje przyznane uprawnienia.

***Permission demand*** (`Demand()`) jest wykorzystywane w celu zapewenienia, by *assembly* posiadające mniejszy zestaw uprawnień **pośrednio nie uzyskało dostępu do zasobu poprzez wywołanie metod z assembly o większych uprawnieniach**.

Wykonanie akcji uprzywilejowanej przez kod nie posiadający odpowiednich uprawnień poprzez wywołanie kodu posiadającego wystarczające uprawnienia określa się mianem ***luring attack***.

57

## Code-Access Security (11)

***Luring attack*** można efektywnie przeciwdziałać jedynie poprzez żądanie konkretnego uprawnienia w górę stosu, aż do rekordu aktywacji znajdującego się na samym spodzie.

Przykładowy proces weryfikacji, czy kod ma uprawnienia do zasobu:
1. *Assembly* A uprawnione jedynie do uruchamiania kodu wywołuje metodę zdefiniowaną w *assembly* B.
   Wywoływana metoda (zdefiniowana w *assembly* B) odczytuje plik.
2. W trakcie odczytywania pliku kod *assembly* B wykorzystuje bibliotekę klas .NET Framework.
3. System bezpieczeństwa .NET Framework sprawdza uprawnienia w górę stosu, aż do pierwszego rekordu aktywacji (`Main()`) weryfikując w ten sposób, czy kod, od którego przyszło żądanie ma dostęp do pliku.
4. System bezpieczeństwa zgłasza wyjątek `SecurityException` ponieważ *assembly* A wywołujące *assembly* B ma uprawnienia jedynie do wykonywania kodu – nie odczytu z pliku.

58

## Code-Access Security (12)

***Permission assert*** zatrzymuje *stack walk* na pewnym poziomie umożliwiając tym samym wykonanie akcji nawet w momencie, kiedy wołające assembly nie posiada uprawnień do wykorzystywanego zasobu.

*Assert* zakłada, że wywoływany kod zapewnia bezpieczny dostęp do zasobu uniemożliwiając jego niewłaściwe wykorzystanie przez assembly uruchamiające.

Jeśli metoda **`Method()`** ma ustawioną akcję *assert* dla uprawnienia **`Permission`** wówczas *stack walk* kończy się sukcesem, o ile *assembly*, w której zdefiniowano metodę **`Method()`** ma przyznane **`Permission`**.

***Assert* przestaje obowiązywać po zakończeniu `Method()`.**

Niewłaściwie użyte ***assert* może otworzyć drogę do *luring attack***, lub w ogólności wprowadzić poważną lukę w systemie bezpieczeństwa.
**Nie można dopuścić, by wołające *assembly* mogło korzystać ze sparametryzowanej metody `Assert()`.**
Nie wolno również nigdy zwracać informacji, do której dostęp wynika z zagwarantowanego (*asserted*) uprawnienia.

59

## Code-Access Security (13)

**Przykładowo:**
Jeśli metoda dopisuje do pliku rejestrującego zdarzenia i gwarantuje **`FileIOPermission`**, nie można dopuścić, by to kod wywołujący określał, do którego konkretnie pliku informacja o zdarzeniu będzie dopisywana.

***Permission deny*** odbiera danemu fragmentowi kodu uprawnienia danego typu.

W momencie przypisania akcji *permission deny* (wywołania **`Deny()`**) **wszystkie żądania danego rodzaju uprawnienia kończą się niepowodzeniem**.
Dzieje się tak nawet w momencie, kiedy dane uprawnienie zostanie przyznane *assembly* w obowiązującej polityce bezpieczeństwa.

Wyjątek stanowi użycie ***assert*** w kodzie wywołującym fragment kodu zgłaszający *permission deny*.
*Assert* unieważnia bowiem *deny* zgłoszone niżej na stosie (w metodach wywołujących – czyli w wyższych adresach pamięci).

60

## Code-Access Security (14)

***Permit only*** umożliwia przyznanie konkretnych uprawnień danemu fragmentowi kodu.
**`PermitOnly`** - podobnie jak **`Deny`** – może być unieważnione przez **`Assert`** zgłoszone w kodzie wywoływanym.

**`Deny`**, **`Assert`** i **`PermitOnly`** to tzw. „akcje nadpisujące" (***overrides***) zmieniające domyślne zachowanie systemu zabezpieczeń.
Jeśli w tym samym rekordzie aktywacji jest wiele *overrides* dotyczących tego samego uprawnienia, wówczas są one przetwarzane w następującej kolejności: (1) **`PermitOnly`**, (2) **`Deny`** i na końcu (3) **`Assert`**.

*Overrides* mogą być unieważnione poprzez wywołanie metod: **`RevertAssert`**, **`RevertDeny`**, **`RevertPermitOnly`**.
Deaktywacja *overrides* dotyczy bieżącego rekordu aktywacji/sekcji stosu (*stack frame*) oraz rekordów znajdujących się poniżej.
Metoda **`RevertAll`** unieważnia wszystkie *overrides* dla danego rodzaju uprawnienia.

61

## Code-Access Security (15)

Z metod **unieważniających *overrides* korzystamy w momencie, gdy chcemy wyłączyć normalne działanie *stack walk* dla określonych fragmentów kodu**.

```
private void AccessSystemDrive () {
    FileIOPermission myPerm = new
      FileIOPermission(FileIOPermissionAccess.Read, "C:\\");

    // do not perform permission check for the below stack
    // frames.
    myPerm.Assert();

    // do some potentially dangerous actions

    // and then cancel the asserted permissions
    CodeAccessPermission.RevertAssert();
}
```

62

## Code-Access Security (16)

Jeśli w trakcie wykonywania *stack walk* system bezpieczeństwa .NET Framework wykryje więcej niż jedno aktywne *override* uprawnienia danego typu (np. *assert i deny*) zgłosi wyjątek.
Przed podmianą *override* innym należy w pierwszej kolejności je unieważnić.

Metody **`Union()`** i **`Intersect()`** umożliwiają odpowiednio wyliczyć sumę oraz iloczyn uprawnień danego rodzaju.

**Przykładowo:**

- uprawnienie A typu **`EnvironmentPermission`** umożliwia dostęp do zmiennych środowiskowych: **`TEMP`** oraz **`SystemRoot`**;
- tymczasem uprawnienie B zezwala na dostęp do zmiennych: **`TEMP`** i **`USERNAME`**.

**`A.Union(B)`** → dostęp do zmiennych: **`TEMP`**, **`SystemRoot`** i **`USERNAME`**
**`A.Intersect(B)`** → dostęp do: **`TEMP`**

Wywołanie **`Demand()`** na obiekcie umożliwiającym dostęp do kilku zmiennych środowiskowych powiedzie się wyłącznie, jeśli zostały przyznane prawa do odczytu wszystkich tych zmiennych.

63

## Code-Access Security (17)

**Imperatywne zgłoszenie akcji bezpieczeństwa** (*security action*): *demand*, *assert*, *deny*, *permit only* polega na:

- jawnym stworzeniu w kodzie instancji danego uprawnienia;
- ustawieniu wymaganych właściwości;
- wywołaniu metody zgłaszającej żądaną akcję: **`Demand()`**, **`Assert()`**, **`Deny()`**, **`PermitOnly()`**.

```
FileIOPermission perm = new FileIOPermission(
   FileIOPermissionAccess.Read, "F:\\TBO\FILE.text" );
try {
   perm.Demand();
   /// perform priviledged action in here
} catch (SecurityException se) { ... }
```

W bibliotece klas .NET Framework wszystkie klasy, które wymagają dostępu do chronionych rodzajów zasobów wywołują **`Demand()`** na instancji odpowiedniego uprawnienia.
Nie ma zatem potrzeby wywoływania **`Demand()`**, chyba że chcemy precyzyjniej określić uprawnienia, np. dostęp do konkretnego pliku.

64

## Code-Access Security (18)

Użycie tzw. **atrybutów bezpieczeństwa** (***security attributes***) przy definicjach klas i metod umożliwia deklaratywne zgłoszenie akcji bezpieczeństwa.

Przy definicji klasy, bądź metody można stosować więcej niż jeden atrybut bezpieczeństwa.

**Przykładowo można dodać:**
- atrybut wymagający (*demand*) konkretnego `EnvironmentPermission`, oraz
- atrybut gwarantujący (*assert*) `FileIOPermission`.

Wszystkie wbudowane w .NET Framework uprawnienia posiadają odpowiadające im atrybuty bezpieczeństwa – klasy rozszerzające klasę abstrakcyjną `CodeAccessSecurityAttribute`.

Atrybuty bezpieczeństwa w trakcie kompilacji trafiają do metadanych (*metadata*) opisywanego przez nich obiektu (klasy, bądź metody) stanowiąc tym samym **część niemodyfikowalnego w trakcie wykonania opisu *assembly***.

65

## Code-Access Security (19)

```
[FileIOPermission(SecurityAction.Demand,
   Read=AccessClass.DIRECTORY)]
class AccessClass {
   const string TEMP = "TEMP",
                SYSTEM_ROOT = "SystemRoot",
                FILE_TXT = DIRECTORY + "\\" + "FILE.txt",
                DIRECTORY = @"F:\TBO";
   ...
   [EnvironmentPermission(SecurityAction.Assert,
      Read=SYSTEM_ROOT)]
   void SystemRootAccess () { ... }

   [FileIOPermission(SecurityAction.Demand,
      Read=FILE_TXT)]
   [EnvironmentPermissmion(Security.Demand, Read=TEMP)]
   void FileTxtAccess () { ... }
}
```

66

## Code-Access Security (20)

***Link demand*** umożliwia sprawowanie kontroli nad tym, jakiego rodzaju *assembly* wywołuje dany kod – tylko ***assembly* spełniające określone kryteria może wywołać dany fragment kodu (metodę, klasę)**.

*Link demand* jest weryfikowane na etapie **kompilacji *just-in-time* przed właściwym uruchomieniem *assembly***.

Użycie *link demand* powoduje, że nie jest wykonywany pełny *stack walk* – weryfikowane jest jedynie *assembly* bezpośrednio wołające dany kod.

*Link demand* znajduje zastosowanie podczas tworzenia **prywatnych *assemblies***, które mogą być wykorzystywane wyłącznie przez kod pochodzący od danego dostawcy.

```
[StrongNameIdentityPermission(SecurityAction.LinkDemand,
   PublicKey = "1234567890ABCD", /// in real life PublicKey is
   /// much much longer
   Name = "AccessAssembly", Version="1.0.0.1")]
class AccessClass { ... }
```

67

## Code-Access Security (21)

Zastosowanie **`SuppressUnmanagedCodeSecurityAttribute()`** umożliwia zamianę uprawnienia wywołania kodu niezarządzanego na *link demand*, co zwiększa wydajność ponieważ odbywa się tylko w trakcie ładowania *assembly* – nie przy każdym wywołaniu metody natywnej.

```
class NativeMethods {
   [SuppressUnmanagedCodeSecurityAttribute()]
   [DllImport("msvcrt.dll")]
   internal static extern int puts(string str);
   [SuppressUnmanagedCodeSecurityAttribute()]
   [DllImport("msvcrt.dll")]
   internal static extern int _flushall();
}
class MainClass {
   [SecurityPermission(SecurityAction.Assert, Flags =
      SecurityPermissionFlag.UnmanagedCode)]
   private static void CallUnmanagedCode () {
      NativeMethods.puts("Hello World!");
      NativeMethods._flushall();
   }
}
```

68

## Code-Access Security (22)

Celem stosowania ***inheritance demand*** jest zapewnienie, by jedynie kod posiadający określone uprawnienia mógł: (1) dziedziczyć z danej klasy, bądź (2) przedefiniowywać (`override`) daną metodę.
Zgłoszenie *inheritance demand* może być podane wyłącznie w formie deklaratywnej.

```
[FileIOPermission(SecurityAction.InheritanceDemand,
   Read = @"F:\TBO")]
class A { ... }
class B : A {
   [FileIOPermission(SecurityAction.InheritanceDemand,
      Read = @"F:\TBO\FILE.txt")]
   public virtual string ReadData () { ... }
}
class C : B {
   /// assembly in which class C is defined must be granted
   /// permission to read file: "F:\TBO\FILE.txt" –
   /// permission to access the directory will not suffice
   public override string ReadData () { ... }
}
```

69

## Code-Access Security (23)

Deklarowane dla *assembly* ***permission request*** określa uprawnienia, które dane *assembly*:
- wymaga, tak, aby mogło być uruchomione;
- może mieć przyznane,
- nie powinno mieć przyznawane.

Niemodyfikowalne w czasie wykonania *permission requests* są przechowywane w metadanych *assembly*.
Zestaw *permission requests* jest weryfikowany w trakcie ładowania *assembly*.

*Permission requests* są podstawowym mechanizmem pozwalającym na zagwarantowanie, że kod nie otrzyma więcej uprawnień, niż jest mu potrzebne do działania.

**W .NET Framework istnieją trzy rodzaje permission requests:**
1. ***minimum permissions*** określające minimalny zestaw uprawnień wymagany dla działania assembly.
   Jeśli polityka bezpieczeństwa nie przyzna minimalnego zestawu uprawnień zostanie zgłoszony wyjątek: `PolicyException`.
   Brak jawnego określenia miminalnych uprawnień oznacza żądanie **Nothing**.

70

## Code-Access Security (24)

**Trzy rodzaje permission requests w .NET Framework (c.d.):**

2. ***optional permissions*** określające uprawnienia, które kod może używać, ale może być wykonywany również bez nich.
   Uprawnienia opcjonalne są udzielane *assembly*, jeśli polityka bezpieczeństwa to umożliwia.
   Jeśli polityka bezpieczeństwa nie przyznaje uprawnień opcjonalnych kod również może być uruchomiony.
   Brak jawnego zadeklarowania `RequestOptional` oznacza, że assembly żąda **FullTrust**.
3. ***refused permissions*** – zestaw uprawnień, które nigdy nie powinny być przyznane *assembly* – nawet jeśli zezwala na to bieżąca polityka bezpieczeństwa.
   Brak deklaracji oznacza, że odrzucany jest zestaw uprawnień: **Nothing**.

*Permission requests* wpływają na ostateczny zestaw uprawnień, jaki otrzymuje assembly, który jest wyliczany następująco:

$$\mathit{FinalGrant} = \mathit{SecurityPolicyGrant} \cap \left( \left( \mathit{RequestMinimum} \cup \mathit{RequestOptional} \right) - \mathit{RequestRefused} \right)$$

71

## Code-Access Security (25)

Podobnie jak miało to miejsce przy deklaracji *security actions* dla danego *assembly* można zadeklarować wiele *permission requests*.

Ograniczanie uprawnień przyznawanych *assembly* za pomocą `RequestOptional` i `RequestRefuse` jest dobrą praktyką, ponieważ redukuje możliwość przeprowadzenia *luring attack*.

Określenia minimalnych uprawnień niezbędnych do wykonania jest również dobrą praktyką, ponieważ użytkownik jest powiadamiany o wszystkich wymaganych uprawnieniach przed uruchomieniem kodu – nie w trakcie, kiedy może wiązać się to z utratą efektów jego pracy.

```
[assembly: FileIOPermission(SecurityAction.RequestMinimum,
   Read = @"F:\TBO\FILE.txt")]
[assembly: SecurityPermission(SecurityAction.RequestOptional,
   UnmanagedCode = true)]
/// request named permission set
[assembly: PermissionSet(SecurityAction.RequestMinimum,
   Name = "Execute")]
```

72

## Role-Based Security (1)

***Role-Based Security*** wykorzystuje informację uzyskaną w trakcie uwierzytelnienia użytkownika w celu ustalenia, z których zasobów może on korzystać.

Fundament ***Role-Based Security*** stanowią dwa pojęcia:
- ***identity*** umożliwiające jednoznaczną identyfikację użytkownika, oraz
- ***principal*** informujące o pełnieniu przez użytkownika danej roli, lub przynależności do grupy.

**CLR** umożliwia realizację autoryzacji w oparciu o *identity* oraz związane z nią role – *principal*.

*Identity* reprezentuje użytkownika, w imieniu którego kod jest wykonywany. *Identity* umożliwia rozróżnienie pomiędzy użytkownikiem uwierzytelnionym oraz anonimowym.

**W ogólności w .NET Framework istnieją trzy rodzaje *identity*:**
- `WindowsIdentity` oparte o uwierzytelnienie dostępne w Windows – umożliwiające „podszywanie się" (*impersonate*) pod innych użytkowników;
- `GenericIdentity` oparte o własną metodę uwierzytelnienia zaimplementowaną w aplikacji;
- implementacja `IIdentity` – własny zestaw informacji dotyczący użytkownika.

73

## Role-Based Security (2)

Autoryzacja w ***Role-Based Security*** jest oparta o *principals* zawierające *identity* oraz informację o rolach pełnionych przez to *identity*.

***Role-Based Security* wspiera trzy rodzaje *principals*:**
- `WindowsPrincipal` reprezentujące użytkownika oraz grupy Windows (role), do których należy;
- `GenericPrincipal` umożliwiające reprezentację grup i użytkowników istniejących niezależnie od użytkowników i grup Windows. `GenericPrincipal` może być wykorzystywane w tworzeniu prostego rozwiązania zapewniającego własne uwierzytelnienie i autoryzację w ramach aplikacji;
- implementacja interfejsu `IPrincipal` zawierająca specyficzną dla aplikacji informację o rolach.

74

## Role-Based Security (3)

**Sposoby tworzenia *principal* wykorzystywanego w trakcie autoryzacji**

Utworzenie *principal* w oparciu o zapytanie Windows o informację dotyczącą zalogowanego użytkownika.

```
/// creation of principal based on querying current Windows
/// account
WindowsIdentity id = WindowsIdentity.GetCurrent();
WindowsPrincipal principal = new WindowsPrincipal(id);
```

Utworzenie *principal* w oparciu o zapytanie skierowane do bieżącego wątku – po uprzednim określeniu rodzaju *principal* związanego z **`AppDomain`**.

```
/// creation of principal based on querying current thread
AppDomain.CurrentDomain.SetPrincipalPolicy(
   PrincipalPolicy.WindowsPrincipal);
WindowsPrincipal principal = Thread.CurrentPrincipal
   as WindowsPrincipal;
```

75

## Role-Based Security (4)

Utworzenie *generic principal* umożliwiającego korzystanie z ***Role-Based Security*** w oparciu o sposób uwierzytelnienia niezależny od Windows – np. w oparciu o dane zgromadzone w bazie danych.

```
GenericIdentity id = new GenericIdentity("user");
string[] roles = {"manager", "teller" };
GenericPrincipal principal = new GenericPrincipal(id, roles);

/// attaching principal to the current thread --- requires
/// ControlPrincipal privilege of SecurityPermission
Thread.CurrentPrincipal = principal

/// or setting principal for new threads
AppDomain.CurrentDomain.SetPrincipal(principal);

/// retrieval of attached principal from the thread
GenericPrincipal gp = Thread.CurrentPrincipal
   as GenericPrincipal;
```

76

## Role-Based Security (5)

Zwykłe porównanie *identity* z *principal* związanego z bieżącym wątkiem z wymaganym *identity* umożliwia kontrolę nad przydzielaniem dostępu do kodu.

```
/// validation based on identity
if (string.Compare(principal.Identity.Name, @"DOMAIN\user")) {
   /// protected code
}
```

Sprawdzenie przynależności do danej roli jest realizowane za pomocą metody **`IsInRole()`** wywołanej na obiekcie principal.

```
/// verification whether the user is a member of Administrators
/// Windows group
if (principal.IsInRole(@"BUILTIN\Administrators")) {
   /// protected code
}

/// as above but using WindowsBuiltInRole enumeration
if (principal.IsInRole(WindowsBuiltInRole.Administrators)) {
   /// protected code
}
```

77

## Role-Based Security (6)

Czasami istnieje konieczność, by aplikacja w celu wykonania danej operacji działała z prawami innymi niż bieżącego użytkownika – np. z prawami administratora.

Mechanizm „podszycia” (*impersonation*) – dostępnego jedynie dla **`WindowsIdentity`** umożliwia czasowe uruchomienie kodu z innymi uprawnieniami, niż posiada użytkownik, w imieniu którego pierwotnie uruchomiono kod.

*Impersonation* opiera się na podmianie żetonu związanego z procesem systemu operacyjnego – *principal* związany z kontekstem wywołania pozostaje niezmienny. W celu uzyskania żetonu konieczne jest wywołanie niezarządzanej metody **`LogonUser`**, która oczekuje podania nazwy użytkownika oraz hasła.

```
[DllImport("advapi32.dll", SetLastError=true)]
public static extern bool LogonUser(String lpszUsername,
   String lpszDomain, String lpszPassword, int dwLogonType,
   int dwLogonProvider, ref IntPtr phToken);
...
LogonUser(..., ref tokenHandle);
WindowsIdentity id = new WindowsIdentity(tokenHandle);
WindowsImpersonationContext context = id.Impersonate();
```

78

## Role-Based Security (7)

Wywołanie **`WindowsImpersonationContext.Undo()`** powoduje usunięcie żetonu z procesu systemu operacyjnego.

Jeśli zachodzi konieczność zastosowania *impersonation* należy pamiętać, by nigdy nie zaszywać haseł do aplikacji w kodzie.
Zapisanie hasła do aplikacji w kodzie *assembly*:

- mogą być stosunkowo łatwo wykryte (np. przy użyciu deasemblera MSIL – **`ildasm.exe`**) , oraz
- uniemożliwiają przestrzeganie polityki haseł – np. dokonywanie zmiany hasłs co pewien określony czas.

Poza tworzeniem własnych mechanizmów autoryzacji w oparciu o instrukcje warunkowe **`if`**, programista może skorzystać z gotowego, dostarczonego przez ***Role-Based Security*** sposobu autoryzacji.
Autoryzacja dostępna w **RBS** jest koncepcyjnie podobna do tego znanej z ***Code-Access Security*** i polega na żądaniu od *principal* związanego z bieżącym wątkiem uprawnień odpowiadających zapisanych w obiekcie **`PrincipalPermission`**.



## Role-Based Security (8)

Podobnie jak w ma to miejsce w **CAS**: (1) tworzymy obiekt reprezentujący uprawnienie, a później (2) wywołujemy na nim metodę odpowiadającą pożądanej akcji.
Autoryzacja oparta o **`PrincipalPermission`** może korzystać wyłącznie z metody **`Demand()`**.

**RBS nie daje możliwości skorzystania z akcji nadpisujących** (*overrides*) – za rodzaj odpowiednika *assert* z CAS można uznać *impersonation* w RBS.

**`PrincipalPermission`** może zawierać co najwyżej jedno *identity* i jedną rolę, zatem tworzenie bardziej złożonych uprawnień wymaga wywołania na obiektach **`PrincipalPermission`** znanych z **CAS** metod: **`Union()`** i **`Intersect()`**.

Podczas wywołania metody **`Demand()`** **CLR** sprawdza, czy *principal* oraz *identity* związane z bieżącym wątkiem pasują do *identity* i roli przechowywanej w obiekcie **`PrincipalPermission`**, na którego rzecz zostało wywołane **`Demand()`**.

Jeśli *identity* i *principal* odpowiadają kryteriom zawartym w **`PrincipalPermission`** wykonywanie jest kontynuowane, w przeciwnym razie zgłaszany jest wyjątek **`SecurityException`**.

## Role-Based Security (9)

W trakcie wykonania metody **`Demand()`** wywołanej na obiekcie **`PrincipalPermission`** nie jest wykonywany *stack walk*, a jedynie weryfikacja własności *principal* związanego z bieżącym wątkiem.
Jeśli sprawdzenie się nie powiedzie ***stack walk* nie będzie w ogóle wykonywany**.

```
/// creates an object of PrincipalPermission that demands
/// the following characteristics from the principal attached
/// to the current thread:
/// identity: "DOMAIN\user", role: "DOMAIN\teller", and that
/// the user must be authenticated
PrincipalPermission permission = new PrincipalPermission(
   @"DOMAIN\user", @"DOMAIN\teller", true);
try {
   permission.Demand();
   /// continue with protected code
} catch (SecurityException se) {
   /// if demand for specific permission fails
}
```



## Role-Based Security (10)

Weryfikacja *principal* związanego z bieżącym wątkiem powiedzie się tylko w momencie, kiedy zarówno *identity*, jak i pełniona rola będą pasowały do *identity* i roli zapisanej w **`PrincipalPermission`**.
Ustawiając jedną z wymienionych właściwości **`PrincipalPermission`** na **`null`** programista może zaznaczyć, że nie interesuje go sprawdzenie danej właściwości *principal*.

Podobnie jak w **CAS** programista może w sposób deklaratywny żądać, by jedynie wątek działający w imieniu użytkownika o danym *identity* i/lub konkretnej roli mógł uruchomić dany fragment kodu.

Deklaratywne żądanie może być umieszczone zarówno na poziomie klas, jak i poszczególnych metod.
O ile deklaracją żądania **`PermissionPrincipal`** zostały opatrzone zarówno definicja klasy, jak i jej metody, deklaracja żądania w metodzie zawsze nadpisuje (*overrides*) żądanie zapisane w jej klasie.

Pominięcie któregoś z elementów deklaracji żądania sprawi, że nie będzie on brany pod uwagę w trakcie weryfikacji *principal* dołączonego do danego wątku.

## Role-Based Security (11)

```
/// Assume that a member of role "manager" simultaneously may
/// play a role of a "teller". Therefore each manager could
/// take advantage of the features of the following class.
/// As regards WindowsPrincipal the above assumption may be
/// implemented by adding Windows user group "manager" to user
/// group "teller".
[PrincipalPermission(SecurityAction.Demand,
   Role = @"DOMAIN\teller", Authenticated = true)]
class ForTellersAndManagers {
   ...
   /// only teller named Joan may invoke PrivateForJoanOnly
   [PrincipalPermission(SecurityAction.Demand,
      Name = @"DOMAIN\Joan", Role = @"DOMAIN\teller",
      Authenticated = true)]
   void PrivateForJoanOnly () { ... }
   /// only a manager may invoke ForManagersExclusively
   [PrincipalPermission(SecurityAction.Demand,
      Role = @"DOMAIN\manager", Authenticated = true)]
   void ForManagersExclusively () { ... }
}
```



## Pliki konfiguracyjne ASP.NET

Ustawienia mechanizmów bezpieczeństwa, z których będą korzystały aplikacje ASP.NET umieszcza się m.in. w plikach konfiguracyjnych będącymi dokumentami XML o ściśle określonej strukturze.

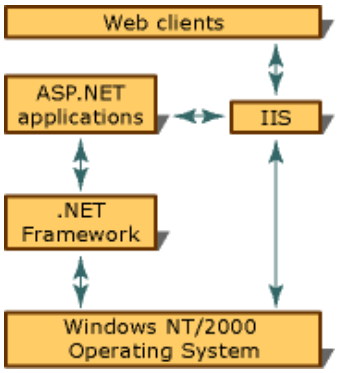
Wyróżniamy dwa rodzaje plików konfiguracyjnych ASP.NET:

1. Znajdujący się w katalogu: **`%WinDir%\Microsoft.NET\Framework\<version>\Config`** plik **`Machine.config`** zawierający ustawienia globalne dla całego serwera.
   Ze względu na możliwość popełnienia błędu, który może mieć wpływ na działanie wszystkich aplikacji zainstalowanych na serwerze, nie jest zalecana bezpośrednia ingerencja w zawartość tego pliku.
2. **`Web.config`** zawierający konfigurację dla katalogu, w którym się znajduje oraz podkatalogów.
   Każda aplikacja ASP.NET musi posiadać przynajmniej jeden plik **`Web.config`**, który jest umieszczony je katalogu głównym.
   Każdy z podkatalogów aplikacji może zawierać własny **`Web.config`** nadpisujący ustawienia bardziej ogólne.
   W momencie dostępu do zasobu ASP.NET określa dla niego konfigurację poprzez hierarchiczne prześledzenie ustawień.

## Uwierzytelnienie w ASP.NET – ogólnie

Uwierzytelnienie w ASP.NET jest zrealizowane przy użyciu tzw. ***authentication providers***, które są odpowiedzialne za weryfikację *credentials* dostarczonych przez użytkownika.

Aplikacje ASP.NET mogą korzystać z kombinacji mechanizmów dostępnych w Internet Information Services (IIS) oraz w samym ASP.NET.
Klienci komunikują się z aplikacją ASP.NET za pośrednictwem IIS, który: (1) dekoduje, oraz (2) opcjonalnie uwierzytelnia żądania klienckie.
IIS jest odpowiedzialny za lokalizację żądanego zasobu – w tym również strony ASP.NET – o ile klient ma do niej dostęp.




Z ASP.NET dostarczane są ***authentication providers*** udostępniające następujące sposoby uwierzytelniania:

- ***IIS authentication (windows authentication provider)***
- ***Forms-based authentication***
- ***Passport authentication***
- ***Anonymous authentication***

85

## Windows Authentication Provider (1)

***windows authentication provider*** – uwierzytelnienie w oparciu o mechanizmy uwierzytelnienia dostępne w Internet Information Services (IIS).

Wszystkie żądania od klienta do aplikacji ASP.NET zanim dotrą do samej aplikacji muszą przejść w pierwszej kolejności przez IIS.
IIS zawsze – przy każdym wspieranym sposobie uwierzytelnienia – zakłada, że zestaw *credentials* zawartych w żądaniu może być zweryfikowany w oparciu o informację zawartą w katalogu użytkowników Windows.
Każdy uwierzytelniany użytkownik **musi posiadać konto w katalogu Windows**.

Jeśli *credentials* **nie odpowiadają istniejącemu i aktywnemu kontu użytkownika** IIS odrzuca żądanie klienta.
Z tego względu ten rodzaj uwierzytelnienia nie jest zalecany dla dużych rozwiązań integrujących różne technologie.

Wybór uwierzytelnienia zintegrowanego z Microsoft Windows oznacza, że ASP.NET jest **zainteresowane wynikiem uwierzytelnienia przeprowadzonego wcześniej przez IIS**, który jest przekazywany do mechanizmów autoryzacji dostępnych w .NET Framework – ***Role-Based Security***.

86

## Windows Authentication Provider (2)

IIS przekazuje do procesu hostującego ASP.NET instancję klasy **`WindowsIdentity`** reprezentującą uwierzytelnionego użytkownika (***identity***). W oparciu o **`WindowsIdentity`** jest tworzony **`WindowsPrincipal`** wiążący **`WindowsIdentity`** z grupami użytkowników Windows, który jest z kolei wiązany z kontekstem aplikacji (**`ApplicationContext`**).

Informacja o uwierzytelnionym użytkowniku jest również przekazywana do samej aplikacji ASP.NET, która może ją skonsumować w odpowiedni dla siebie sposób – np. wiążąc z **`WindowsIdentity`** instancję innej klasy implementującej **`IPrincipal`**.

```
<configuration>
   <system.web> <authentication mode="Windows"> </system.web>
</configuration>
```

IIS udostępnia następujące sposoby uwierzytelnienia:

- ***basic authentication***
- ***digest authentication***
- ***Integrated Windows authentication***
- ***client certificate authentication***
- ***anonymous access***

87

## Windows Authentication Provider (3)

***Basic authentication*** jest sposobem uwierzytelnienia opartym o mechanizm udostępniany przez HTTP, w którym serwer wysyła do klienta żądanie (*challenge*), w odpowiedzi na które oczekuje podania przez klienta nazwy użytkownika i hasła.

Dane uwierzytelniające (*credentials*) są przesyłane od klienta do serwera otwartym tekstem (zakodowane jedynie w Base64) w nagłówku żądania..
Z tego względu w celu zabezpieczenia procesu uwierzytelnienia należy zastosować tunelowanie ruchu sieciowego w SSL/TLS/IPSec.

Scenariusz realizacji uwierzytelnienia w oparciu o skrót (***digest authentication***) jest zbliżony do *basic authentication* i również jest oparty o dostępny w HTTP mechanizm uwierzytelnienia ***challenge/response***.

W odróżnieniu do uwierzytelnienia podstawowego dane uwierzytelniające są przesyłane przez sieć w postaci skrótu (*hash*) (zazwyczaj wyliczanego przy użyciu MD5).
Serwer wysyła do klienta losowy ciąg oktetów (***nonce value***).
*Nonce value* razem z m.in. nazwą użytkownika, hasłem, URI aplikacji oraz *realm* (domena aplikacyjna w HTTP) będzie stanowił dla klienta podstawę do wyliczenia odpowiedzi, którą z powrotem prześle do serwera.

88

## Windows Authentication Provider (4)

Ponieważ po stronie IIS wykonywane są te same wyliczenia, co po stronie klienckiej, wymagane jest by **hasła w katalogu użytkowników Windows były zaszyfrowane w sposób odwracalny**.
Co prawda w RFC 2617 znajduje się opis, w jaki sposób można zrealizować *digest authentication* przechowując jedynie skróty haseł, ale **ta funkcjonalność nie została zaimplementowana w IIS**.
O stopniu bezpieczeństwa tego rodzaju uwierzytelnienia decyduje **na ile rzeczywiście losowa jest *nonce value***.

W przypadku **uwierzytelnienia zintegrowanego z Windows (*windows integrated authentication*)** IIS uzyskuje *credentials* z systemu operacyjnego stacji, na której działa oprogramowania klienckie.
*Credentials* są przesyłane do IIS za pomocą **protokołów uwierzytelnienia wykorzystywanych w Windows**: NTLM (***challenge/response***) lub Kerberos v5.

Pakiet uwierzytelnienia NTLM (NT LAN Manager) dostarczany z Microsoft Windows zawiera w rzeczywistości implementacje trzech protokołów uwierzytelnienia: LAN Manager, oraz jego wersje rozwojowe – NT LAN Manager v1 (Windows NT 4.0), oraz NT LAN Manager v2 (Windows NT 4.0 SP4).

89

## Windows Authentication Provider (5)

Uwierzytelnienie klienta w oparciu o dostępny w Windows protokół ***challenge/response*** (NTLM v2) ma przebieg bardzo zbliżony do uwierzytelnienia wykorzystującego znane z HTTP *digest authentication*.

Kiedy klient próbuje uzyskać dostęp do zasobu umieszczonego na serwerze, serwer wysyła do klienta losowy ciąg oktetów – *challenge*.
Klient (przeglądarka) w oparciu o *challenge* tworzy odpowiedź do serwera poprzez wyliczenie skrótu od *challenge* połączonego z hasłem, który jest przesyłany do serwera.

Po stronie serwera wykonywana jest dokładnie ta sama operacja.
Jeśli wynikowy ciąg znaków odpowiada zawartości komunikatu przesłanego przez klienta, tożsamość klienta jest pozytywnie zweryfikowana.

**Kerberos v5 jest podstawowym protokołem uwierzytelnienia w domenach W2K** umożliwiającym serwerowi i klientowi wzajemną weryfikację tożsamości drugiej strony.
W oparciu o Kerberos v5 została w ramach domeny opartej o Active Directory zrealizowana koncepcja uwierzytelnienia jednorazowego (*single sign-on*).

90

## Windows Authentication Provider (6)

**Ogólny scenariusz uwierzytelnienia w oparciu o Kerberos v5:**

1. Użytkownik stacji klienckiej uwierzytelnia się przed Key Distribution Center (KDC) uruchomionym na każdym kontrolerze domeny opartej o Active Directory.
2. KDC wystawia klientowi tzw. bilet uprawniający do przyznawania biletów do usług (***ticket-granting ticket*** – TGT).
   TGT jest wykorzystywany w imieniu użytkownika przez oprogramowanie klienckie w trakcie dostępu do usługi przyznającej dostęp do usług sieciowych (***ticket-granting service*** – TGS).
3. TGS na żądanie klienta wystawia tzw. bilety do usług (***service tickets*** – TS).
4. TS jest wykorzystywany przez oprogramowanie klienckie (np. przeglądarkę) w trakcie uwierzytelnienia przed usługą (serwerem, na którym został uruchomiony IIS).
5. Jeśli w TS zawarto prośbę o wzajemne uwierzytelnienie strona serwerowa szyfruje przy użyciu klucza sesyjnego informację (znacznik czasu) wcześniej zaszyfrowaną jej kluczem publicznym przez KDC.

Przebieg opisanego procesu – poza wprowadzeniem początkowych *credentials* wymaganych dla uwierzytelnienia przed KDC – jest niewidoczny dla użytkownika.

91

## Windows Authentication Provider (7)

Realizacja wymiany *credentials* w oparciu o protokoły *challenge/response* wykorzystywane w uwierzytelnieniu Windows lub Kerberos v5 sprawia, że **ten rodzaj uwierzytelnienia ma ograniczone zastosowanie**.

*Integrated Windows authentication* powinno być stosowane wyłącznie w intranecie, gdzie pakiety nie muszą przekraczać ściany ogniowej, bądź *proxy* – chyba, że mamy do czynienia z implementacją VPN.

**Uwierzytelnienie w oparciu o certyfikat X.509 klienta (*client certificate authentication*)** jest oparte o tryb uwierzytelnienia klienckiej (bądź klienckiej i serwerowej) – dostępny w protokole SSL/TLS.

Przedstawianie własnego certyfikatu stronie przeciwnej w celu potwierdzenia własnej tożsamości jest opcjonalną częścią tzw. *handshake protocol* SSL/TLS.

Najczęściej wykorzystywanym scenariuszem *handshake* jest uwierzytelnienie strony serwerowej przez stronę kliencką.

W trybie wykorzystywanym w *client certificate authentication* serwer oczekuje o strony klienckiej okazania ważnego certyfikatu X.509 wystawionego przez zaufane centrum certyfikacji (*certificate authority* – CA).

92

## Windows Authentication Provider (8)

Po pomyślnej weryfikacji, czy certyfikat został wystawiony przez zaufane CA oraz ważności certyfikatu – w oparciu o datę wygaśnięcia (*expiry date*) oraz *revocation list* – serwer wydobywa z certyfikatu dane identyfikujące użytkownika: *Common Name* (CN).
Certyfikat może być następnie zmapowany na konto użytkownika – istnieje również możliwość zmapowania wielu certyfikatów na to samo konto.

Uwierzytelnienie oparte o certyfikat kliencki X.509 jest uważane za **najbezpieczniejszy rodzaj uwierzytelnienia spośród wspieranych przez IIS**.

Podobnie jak w każdym scenariuszu uwierzytelnienia opartym o certyfikat X.509 podstawą zaufania jest fakt legitymowania się przez drugą stronę (*peer*) certyfikatem wystawionym przez zaufane ***Certificate Authority*** (CA).
Z tego względu wiąże się:

- albo z wysokimi kosztami wystawienia użytkownikom certyfikatów przez znane centrum certyfikacji (Verisign, RSASecurity, Thawte, ...);
- albo niejednokrotnie niemałymi kosztami utrzymania własnego centrum certyfikacji.



## Windows Authentication Provider (9)

**Uwierzytelnienie anonimowe (*anonymous authentication*)** jest wykorzystywane w momencie, kiedy nie ma potrzeby uwierzytelniania strony klienckiej przez IIS.

IIS akceptuje przychodzące żądania i przekazuje je bezpośrednio do ASP.NET.
Sama aplikacja jest wykonywana w kontekście specjalnego lokalnego użytkownika (domyślnie **`IUSR_<machinename>`** – kontem użytkownika należącego do grupy **`Guests`**).

W przypadku ustawienia uwierzytelnienia anonimowego warto zastanowić się, czy w konfiguracji aplikacji ASP.NET (bądź całego serwera – **`Machine.config`**) nie powinno być w sposób jawny określone konto, w kontekście którego powinien być uruchomiony proces ASP.NET (*worker process* lub inaczej *host process*).

```
<configuration>
   <system.web>
      <authentication mode="Windows">
      <processModel enable="true" username="DOMAIN\user"
         password="password"/>
   </system.web>
</configuration>
```

## Windows Authentication Provider (10)

O tym, z którego mechanizmu (bądź mechanizmów) uwierzytelnienia spośród dostarczanych przez IIS będzie korzystała określona aplikacja ASP.NET decydują ustawienia IIS, które można zmodyfikować za pomocą narzędzia Internet Service Manager.

Jeśli wybranych zostanie jednocześnie kilka mechanizmów uwierzytelnienia, IIS będzie próbował kolejno uwierzytelnić użytkownika w oparciu o każdy z wybranych mechanizmów w kolejności od najbardziej do najmniej bezpiecznego.

Przyjmuje się następującą kolejność w zakresie zaufania do przedstawionych metod uwierzytelniania wspieranych przez IIS:
1. *client certificate authentication*,
2. *integrated Windows authentication*,
3. *digest authentication*,
4. *basic authentication*,
5. *anonymous authentication*.

95

## Impersonation (1)

Aplikacje ASP.NET są wykonywane w ramach uruchomionych w systemie operacyjnym procesów ASP.NET – tzw. procesów hostujących (*host processes*, *work processes*) ASP.NET.
W Microsoft Windows w ramach każdego procesu systemowego może być uruchomionych wiele wątków.

Mechanizm „wcielenia” (*impersonation*) umożliwia **wątkowki pracę w innym kontekście bezpieczeństwa niż proces, w ramach którego został uruchomiony**.

Impersonation jest wygodnym sposobem autoryzacji w oparciu o mechanizmy istniejące w Windows.
Korzystając z *impersonation* można zawęzić, bądź rozszerzyć uprawnienia aplikacji.

„Wcielenie” umożliwia aplikacji ASP.NET działanie w kontekście bezpieczeństwa konta uwierzytelnionego użytkownika – zamiast domyślnego konta ASP.NET – w celu odpowiedniego ograniczenia jego praw dostępu.

Uprawnienia są ustalane w oparciu o listy kontroli dostępu (*access control lists*) przypisane do poszczególnych zasobów (plików, katalogów przechowywanych w systemie plików NTFS).

96

## Impersonation (2)

Przypuśćmy, że do każdego pliku zawierającego informację poufną dołączono ACL. W celu zabezpieczenia przed nieuprawnionym dostępem aplikacja ASP.NET może być wykonywana w kontekście bezpieczeństwa uwierzytelnionego użytkownika.

Mechanizm ***impersonation* jest dostępny wyłącznie w przypadku *integrated Windows authentication***.

IIS przekazuje aplikacji żeton zawierający informację o uwierzytelnionym, bądź nieuwierzytelnionym użytkowniku.
Na podstawie żetonu proces hostujący ASP.NET zezwala, bądź nie zabrania dostępu w oparciu o ACL związany z zasobem.

*Impersonation* może być skonfigurowane na stałe w **`Web.config`**, jak również ustawione dynamicznie w aplikacji poprzez użycie metody **`WindowsIdentity.Impersonate()`**, cofnięcie *impersonation* odbywa się przy użyciu **`WindowImpersonationContext.Undo()`**.

```
<configuration>
   <system.web><identity impersonate="true"/></system.web>
</configuration>
```

97

## Impersonation (3)

Chęć korzystania z *impersonation* musi być zakomunikowana w sposób jawny – **domyślnie mechanizm „wcielenia" jest wyłączony na poziomie `Machine.config`**.

Włączenie *impersonation* w przypadku anonimowego uwierzytelnienia IIS spowoduje uruchomienie wątku aplikacji w kontekście bezpieczeństwa użytkownika przypisanego do dostępu anonimowego IIS – domyślnie **`IUSR_<machinename>`**.

Konfiguracja aplikacji ASP.NET umożliwia również jawne określenie, w którego użytkownika powinna się wcielić się aplikacja, jeśli nie powinien być to użytkownik uwierzytelniony.

Należy pamiętać, że w takim wypadku zarówno nazwa użytkownika, jak i hasło są podane otwartym tekstem.

```
<configuration>
   <system.web>
      <identity impersonate="true" userName="MACHINE\user"
         password="p@$$w*rd"/>
   </system.web>
</configuration>
```

98

# Impersonation (4)

Istnieje również możliwość przechowywania zaszyfrowanych wartości nazwy użytkownika i hasła w rejestrze Windows.

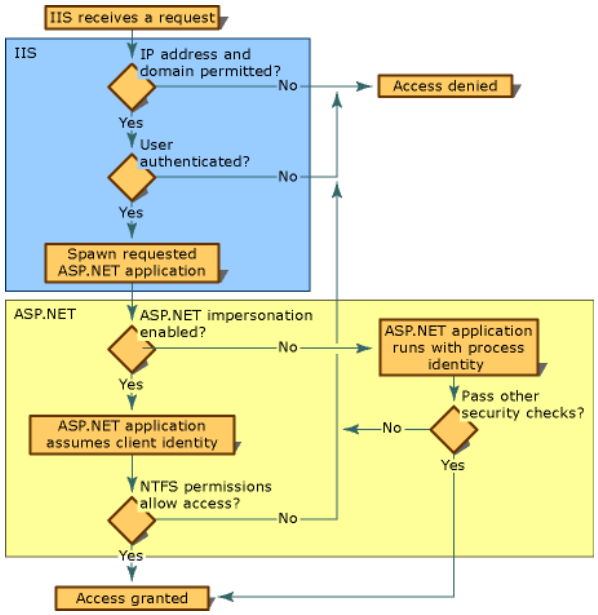
```
<configuration><system.web>
  <identity impersonate="true"
   userName="registry:HKLM\Software\AspNetIdentity,Name"
   password="registry:HKLM\Software\AspNetIdentity,Password"/>
</system.web></configuration>
```

Wartości przechowujące *credentials* muszą być zapisane w formacie **`REG_BINARY`** i być rezultatem wywołania funkcji **`cryptProtectData`** z API Windows.
W celu umieszczenia *credentials* w rejestrze Windows należy posłużyć się aplikacją: ASP.NET Set Registry (**`Aspnet_setreg.exe`**).



# Windows Authentication Provider & Impersonation

**Ogólny schemat przebiegu uwierzytelnienia w oparciu o mechanizmy dostępne w IIS (*windows authentication provider*)**



# Form-Based Authentication (1)

W celu ochrony wybranych stron przed nieuprawnionym dostępem programista może posłużyć się mechanizmem **uwierzytelnienia opartym o własną stronę logowania – własny formularz** (***forms-based authentication***).

Podstawą *forms-based authentication* jest dostępny w HTTP mechanizm przekierowywania.

Użytkownicy nieuwierzytelnieni przy próbie dostępu do chronionych zasobów są kierowani do specjalnie wyznaczonej strony zawierającej formularz umożliwiający wprowadzenie *credentials*, które następnie są przekazywane do aplikacji ASP.NET.
W przypadku pomyślnego uwierzytelnienia aplikacja wystawia **ciasteczko uwierzytelniające (*authentication cookie*)**, które jest kojarzone z sesją użytkownika, po czym użytkownik jest przekierowywany z powrotem do **`RETURNURL`**.
*Authentication cookie* jest przekazywane do aplikacji w nagłówkach kolejnych żądań HTTP.
*Authentication cookie* nie musi być utrwalane w przeglądarce.



# Form-Based Authentication (2)

Chęć korzystania z uwierzytelnienia opartego o formularze należy jawnie zaznaczyć w pliku konfiguracyjnym **`Web.config`** aplikacji ASP.NET.

```
<configuration>
   <system.web>
      <authentication mode="Forms">
         <forms loginUrl = "login.aspx"/>
      </authentication>
   </system.web>
</configuration>
```

Strona logowania to zazwyczaj prosta strona zawierająca dwa pola tekstowe służące do wprowadzania nazwy użytkownika oraz hasła.
Zadaniem strony logowania jest weryfikacja dostarczonych przez użytkownika *credentials* z wykorzystywanym przez aplikację katalogiem użytkowników: plikiem konfiguracyjnym, bazą danych, Active Directory, katalogiem LDAP, itp.

*Forms-based authentication* wiąże się zazwyczaj z ustawieniem w IIS uwierzytelnienia anonimowego (*allow anonymous*).
W przeciwnym *forms-based authentication* będzie przeprowadzone dopiero po pomyślnym uwierzytelnieniu użytkowników przez mechanizmy dostępne w IIS.

## Form-Based Authentication (3)

Oczywiście nie musi tak być jeśli chcemy skorzystać z dwóch mechanizmów uwierzytelnienia jednocześnie (*two-factor authentication*) – np. zanim użytkownik dostarczy *credentials* będzie musiał przedstawić identyfikujący go certyfikat X.509 wystawiony przez zaufane CA.
Należy jednak pamiętać, że w takim wypadku nawet jeśli IIS będzie wymagał od użytkownika okazania certyfikatu X.509, **informacja zawarta w certyfikacie nie będzie dostępna dla aplikacji ASP.NET poprzez standardowe mechanizmy**.

W trakcie procesu uwierzytelnienia opartego o formularze *credentials* są przesyłane otwartym tekstem – należy zatem zadbać o tunelowanie komunikacji uwierzytelniającej w SSL/TLS lub IPSec.

W przypadku uwierzytelnienia opartego o formularze istnieje również możliwość umieszczenia w pliku konfiguracyjnym **`Web.config`** listy zawierającej *credentials*, które mogą posłużyć do weryfikacji tożsamości użytkowników.



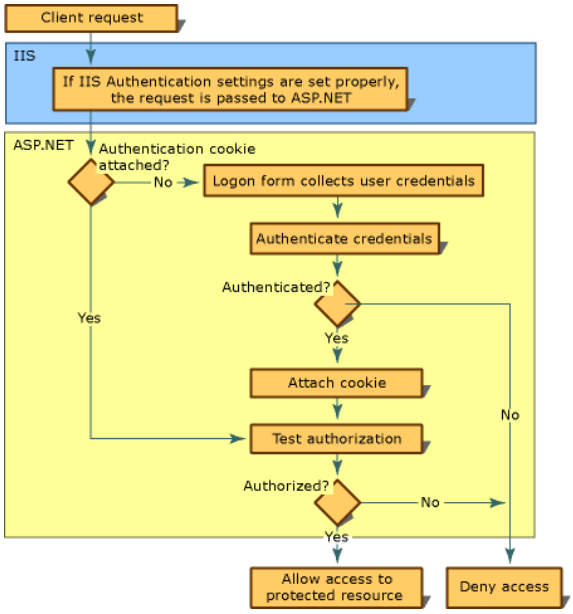
## Form-Based Authentication (4)

```
<authentication mode="Forms">
  <forms name="SavingsPlan" loginUrl="/logon.aspx">
    <credentials passwordFormat="SHA1">
      <user name="Kim"
          password="07B7F3EE06F278DB966BE960E7CBBD103DF30CA6"/>
      <user name="John"
          password="BA56E5E0366D003E98EA1C7F04ABF8FCB3753889"/>
    </credentials>
  </forms>
</authentication>
```

Pomimo, że hasła można podawać w postaci skrótów MD5 lub SHA-1, należy pamiętać, że tego rodzaju uwierzytelnienie jest dobre jedynie na czas testowania aplikacji.

# Form-Based Authentication (5)

**Ogólny schemat przebiegu uwierzytelnienia w oparciu o formularze (*forms-based authentication*)**




105

# Passport-Based Authentication (1)

***Passport-based authentication*** korzysta z Microsoft Passport® – scentralizowanej usługi uwierzytelnienia dostarczonej przez Microsoft umożliwiającej realizację ***single sign-on*** – jednorazowego uwierzytelniania – w ramach witryn korzystających z tej usługi.

Celem usługi webowej (*web service*) Microsoft Passport jest dostarczenie jednego wspólnego mechanizmu uwierzytelnienia dla wszystkich witryn korzystających z usługi Passport.

Usługa Microsoft Passport stanowi odpowiedź na problem tzw. „rozmnażania się" danych uwierzytelniających (***credentials proliferation***).
Za pomocą jednego zestawu *credentials* użytkownik może uzyskać dostęp do wszystkich witryn zarejestrowanych w Microsoft Passport.

Uwierzytelnienie w oparciu o Microsoft Passport wymaga, by:

- witryna umożliwiała korzystanie z tej usługi;
- osoby korzystające z aplikacji ASP.NET posiadały konta użytkowników w ramach usługi .NET Passport.

106

## Passport-Based Authentication (2)

W momencie dostępu do zasobu wymagającego uwierzytelnienia strona, do której użytkownik chce się dostać wyszukuje ciastka (*cookie*) Microsoft Passport w nagłówku żądania HTTP.

Podobnie jak ma to miejsce w przypadku *forms-based authentication* jeśli strona nie znajdzie ciasteczka Microsoft Passport w nagłówku żądania, użytkownik zostanie przekierowany do witryny .NET Passport.

Po pomyślnym uwierzytelnieniu użytkownik jest z powrotem przekierowywany do witryny chronionej przez Passport.

Podpięcie własnej witryny do Microsoft Passport wiąże się w pierwszej kolejności z jej zarejestrowaniem usłudze Passport.
Należy także ściągnąć ze strony Microsoft tzw. Passport SDK zawierające *assemblies* umożliwiające wykorzystanie usługi Passport we własnych rozwiązaniach.

Od strony prawnej rejestracja witryny w .NET Passport wymaga podpisania umowy licencyjnej z firmą Microsoft na dostarczanie usługi uwierzytelnienia.
**Wykorzystywanie usługi Microsoft Passport w procesie uwierzytelniania jest odpłatne**.



## Passport-Based Authentication (3)

Korzystanie z uwierzytelnienia Passport wymaga jawnego zadeklarowania w pliku konfiguracyjnym **`Web.config`**.

```
<configuration>
   <system.web>
      <authentication mode="Passport">
         <passport redirectUrl="login.aspx"/>
      </authentication>
   </system.web>
</configuration>
```

Aplikacja korzystająca z uwierzytelnienia .NET Passport może wykorzystywać instancje klasy **`PassportIdentity`**, które poza *credentials* oraz danymi osobowymi użytkownika mogą zawierać również profil umożliwiający przechowywanie m.in. numerów kart kredytowych, adresów korespondencyjnych, itp.

Informacje zgromadzone w profilu Passport mogą posłużyć twórcom aplikacji do **odpowiedniego wypełnienia formularzy (!)** – np. w trakcie robienia przez użytkownika zakupów w sklepie internetowym!

## Passport-Based Authentication (4)

Na czas tworzenia i testowania rozwiązania można utworzyć tzw. konto PREP (*pre-production account*) – zarówno dla witryny, jak i dla jej użytkowników.
Korzystanie ze środowiska PREP stanowi dla Microsoft potwierdzenie podejmowania wysiłków na rzecz oparcia uwierzytelnienia o Passport.

Środowisko PREP jest środowiskiem całkowicie oddzielonym od produkcyjnego środowiska posiadające osobny rejestr witryn oraz zbiór kont użytkowników.
W przypadku korzystania z wersji produkcyjnej .NET Passport użytkownik jest przekierowywany do **`http://login.passport.com`**, natomiast w przypadku środowiska PREP – **`http://current-login.passporttest.com`**.

Organizacje raczej są niezbyt skłonne oddawać kontrolę nad tak ważnym procesem jak uwierzytelnienie podmiotom trzecim.
Oznacza to bowiem uzależnienie – nie tylko od dostawcy usługi, ale również od jakości oraz dostępności samej usługi.
W ogólności uwierzytelnienie oparte o Passport zaleca się jako alternatywne w stosunku do podstawowego, nad którym dana organizacja sprawuje pełną kontrolę.



## Anonymous authentication

Konfiguracja aplikacji ASP.NET umożliwia również całkowitą rezygnację z uwierzytelnienia (*anonymous authentication*), bądź korzystanie z własnego mechanizmu uwierzytelnienia dostosowanego do konkretnych potrzeb użytkownika (*custom authentication*).

Przykładem własnego schematu uwierzytelnienia może być filtr wykorzystujący Internet Server Application Programming Interface (ISAPI).
Nasz własny filtr realizowałby własny mechanizm uwierzytelnienia oraz przykładowo tworzył obiekt klasy **`GenericPrincipal`**.

```
<configuration>
   <system.web>
      <authentication mode="None"/>
   </system.web>
</configuration>
```

Oczywiście konfiguracja określająca brak uwierzytelnienia nie przesądza, że nie będzie wykonywane uwierzytelnienie po stronie IIS – tyle, że aplikacja ASP.NET nie będzie zainteresowana wykorzystaniem rezultatów tego uwierzytelnienia.

# Autoryzacja w ASP.NET (1)

Poza ***Code-Access Security*** oraz ***Role-Based Security*** twórca aplikacji ASP.NET może dodatkowo skorzystać z dwóch mechanizmów ograniczających uprawnienia użytkowników standardowo udostępnianych przez ASP.NET – czyli:

- ***file authorization***, oraz
- ***URL authorization***.

***File authorization*** działa w oparciu o weryfikację uprawnień aplikacji ASP.NET działającej w kontekście danego konta użytkownika i jest mechanizmem dostępnym automatycznie w momencie uaktywnienia *integrated Windows authentication*.

Przy próbie dostępu do zasobów chronionych IIS w pierwszej kolejności uwierzytelnia użytkownika – jeśli nie został wcześniej uwierzytelniony.
Po przeprowadzeniu uwierzytelnienia uprawnienia wątku procesu hostującego ASP.NET związanego z sesją użytkownika są weryfikowane z prawami zapisanymi w ACL towarzyszącym obiektowi: plikowi, katalogowi.

Jeśli aplikacja nie wykorzystuje *integrated Windows authentication* jedynym standardowo dostępnym mechanizmem kontroli dostępu do zasobów w aplikacjach ASP.NET jest ***URL authorization***.

111

# Autoryzacja w ASP.NET (2)

***URL authorization*** decyduje o przyznaniu, bądź nieprzyznaniu dostępu do zasobu chronionego w oparciu o ***identity***, **role** przypisane aplikacji ASP.NET w trakcie uwierzytelnienia, a także rodzaj żądania HTTP, które przyszło od klienta: **`GET`** lub **`POST`**.

Konfiguracja autoryzacji opartej o URL jest całkowicie zapisywana w plikach konfiguracyjnych aplikacji.

Składnia pliku **`Web.config`** umożliwia:

- określenie ograniczeń w dostępie do zasobów (plików, katalogów) dla indywidualnych użytkowników, bądź też grup użytkowników (ról), jak również rodzajów żądań klienckich (**`GET`** lub **`POST`**);
- określenie, czy ograniczenia w dostępie do zasobów dotyczy całości aplikacji, czy też wybranych zasobów: stron (plików), lub podkatalogów aplikacji.

W trakcie definicji uprawnień można posługiwać się symbolami wieloznacznymi (*wildcards*):

- **`"*"`** – oznacza każdego użytkownika, bądź każdą grupę;
- **`"?"`** – oznacza użytkownika anonimowego.

112

## Autoryzacja w ASP.NET (3)

```
<configuration> <system.web>
   <authorization>
      <allow users="DOMAIN\user1, DOMAIN\user2"/>
      <deny users="*"/>
   </authorization>
</system.web> </configuration>
```

```
<authorization>
   <deny users="?"/>
</authorization>
```

W trakcie określania uprawnień dla konkretnych użytkowników bądź grup użytkowników **należy pamiętać, by zabronić dostępu wszystkim pozostałym**.

Ewaluacja uprawnień odbywa się w kolejności podanej w pliku konfiguracyjnym. ASP.NET przetwarza po kolei wszystkie uprawnienia dopóki nie natrafi na element, do którego będzie pasował żeton użytkownika dołączony do wątku obsługującego sesję.

Z tego względu kluczowe znaczenie ma **kolejność deklaracji elementów wchodzących w skład bloku `<authorization>`**.

113

## Autoryzacja w ASP.NET (4)

```
<authorization>
   <deny users="*"/> <!-- deny access to all users -->
   <allow roles="DOMAIN\Administrators"/>
   <allow users="DOMAIN\user1, DOMAIN\user2"/>
</authorization>
```

Użycie atrybutu **VERB** umożliwia ograniczenie dostępu w oparciu o rodzaj żądania HTTP.

```
<authorization>
   <allow VERB="POST" users="user1, user2"/>
   <deny VERB="POST" users="*"/>
   <allow VERB="GET" users="*"/>
</authorization>
```

Domyślna konfiguracja autoryzacji zapisana w konfiguracji całego serwera IIS (**`Machine.config`**) umożliwia nieograniczony dostęp do wszystkich zasobów.

```
<authorization>
   <allow users="*"/>
</authorization>
```

114

## Autoryzacja w ASP.NET (5)

Zabezpieczenie przed nieuprawnionym dostępem powinno obejmować również **przypisanie w konfiguracji IIS odpowiednich uprawnień do zasobów znajdujących się w wirtualnym katalogu**.

Najbardziej oczywistym i jednocześnie najprostszym sposobem określenia uprawnień do zasobów umieszczonych w katalogu wirtualnym są listy kontroli dostępu (ACL) w NTFS.

Przy stosowaniu kontroli dostępu opartej o ACL zalecane jest raczej przypisywanie ACL do całych katalogów niż pojedynczych plików.
Pliki dodawane do danego katalogu dziedziczą z niego uprawnienia, z tego względu stosowanie ACL do katalogów zmniejsza prawdopodobieństwo popełnienia pomyłki.

Niezależnie od ACL możliwe jest przydzielenie katalogowi udostępnianemu przez HTTP (*web folder*) uprawnień specyficznych dla IIS, czyli:
- **uprawnień do katalogu wirtualnego (*virtual directory permissions*)**, oraz
- **uprawnień do wykonywania (*execute permissions*)**.

115

## Autoryzacja w ASP.NET (6)

***Virtual directory permissions*** obejmują:
- ***Read*** – umożliwia użytkownikowi przeglądanie zawartości folderów oraz plików. Jest to uprawnienie przypisywane domyślnie;
- ***Write*** – użytkownik ma możliwość zmiany zawartości pliku bądź katalogu;
- ***Script source access*** – zezwala użytkownikowi na dostęp do plików źródłowych aplikacji.
  Połączenie tego uprawnienia z *Read* lub *Write* powoduje, że użytkownik ma wyłącznie prawa do odczytu, bądź zapisu plików źródłowych skryptów;
- ***Directory browsing*** – użytkownik posiada uprawnienia do przeglądania listy plików znajdujących się w katalogu;
- ***Log visits*** – tworzy wpis do logu rejestrującego odwiedziny użytkownika.

***Execute permissions*** określają poziom, na którym użytkownik może uruchamiać aplikacje udostępniane przez IIS – dostępne poziomy to:
- ***None*** – użytkownik nie może uruchomić skryptów (czyli np. aplikacji ASP.NET), ani plików wykonywalnych;
- ***Scripts only*** – użytkownik może uruchamiać wyłącznie skrypty;
- ***Scripts and Executables*** – użytkownik może uruchamiać zarówno skrypty, jak i pliki wykonywalne.

116

## Autoryzacja w ASP.NET (7)

Ostatnim miejscem, w którym – standardowo – można zdefiniować prawa dostępu do zasobów znajdujących się w danym katalogu wirtualnym jest plik konfiguracyjny aplikacji **`Web.config`**.

```
<configuration>
   <location path="Default Web Site/Application">
      <system.web>
         <identity impersonate="true" userName="MACHINE\user"
            password="p@$$w*rd"/>
      </system.web>
   </location>
</configuration>
```



## Isolated Storage (1)

Bardzo często zdarza się, że aplikacja w trakcie wykonywania musi utrwalać na dysku własne ustawienia, dane użytkownika, lub innego różnego rodzaju informacje.
Zazwyczaj do przechowywania tego rodzaju informacji stosuje się specjalnie wyznaczone miejsca w systemie plików, lub miejsca w których zapisuje się dane tymczasowe – przykładowo katalog **`%TEMP%`** w Windows, lub **`/tmp`** w systemach Unix-owych.

Czasami ścieżka do tego rodzaju miejsc jest na trwałe zaszyta w kodzie aplikacji (*hard-coded*), co oczywiście znacząco zmniejsza elastyczność w zakresie konfiguracji i utrudnia administrację.
W bardziej przemyślanych rozwiązaniach ścieżka do katalogu przechowującego informację tymczasową odczytywana jest albo z pliku konfiguracyjnego, albo z odpowiednich wpisów w rejestrze Windows.

**Izolowane przechowywanie (*isolated storage*)** plików stanowi realizację koncepcji wirtualnego systemu plików, który jest związany z konkretnym *assembly* w oparciu informację towarzyszącą *assembly* w trakcie wykonywania.

Przykładowo programista może przypisać *isolated storage* do *assembly* w oparciu o miejsce jego pochodzenia (*origin*), bądź informację o jego wydawcy (*publisher*).

## Isolated Storage (2)

Z reguły *isolated storage* jest umieszczane po stronie stacji klienckiej, ale w przypadkach użytkowników „przemieszczających się” (*roaming users*) bardziej sensowne jest zlokalizowanie *isolated storage* po stronie serwerowej.

Wykorzystywanie *isolated storage* w kodzie aplikacji jest uzasadnione zwłaszcza w sytuacji, kiedy wiadomo, że aplikacja będzie uruchamiana z ograniczonym zestawem uprawnień – zwłaszcza w zakresie dostępu do systemu plików.
Z drugiej zaś strony wiadomo, że funkcjonalność aplikacji, bądź wygoda użytkowania wymaga, by aplikacja miała możliwość utrwalania części danych – tak jak czynią to aplikacje posiadające dostęp do systemu plików.

Opisane ograniczenia dotyczą zwłaszcza aplikacji uruchamianych z zestawami uprawnień przypisanymi do stref: *Internet* oraz *intranet*.

W przedstawionym przypadku za wyborem *isolated storage* przemawiają dwie cechy funkcjonalne tego mechanizmu:
1. *Isolated storage* zapewnia bezpieczny oraz w pełni nadzorowany dostęp do plików bez bezpośredniego przyznawania uprawnień do systemu plików.
2. Istnieje możliwość odebrania innym aplikacjom dostępu do *isolated storage* danej aplikacji w celu zabezpieczenia przed nieuprawnionym dostępem do zasobów tej aplikacji.



## Isolated Storage (3)

*Isolated storage* może stanowić również dobry wybór, nawet wtedy, gdy mamy do czynienia z aplikacją uruchamianą z pełnym zestawem uprawnień (*fully trusted*).
Korzystanie z *isolated storage* sprawia, że nie ma potrzeby zaszywania w kodzie ścieżek do zasobów, bądź plików konfiguracyjnych.

Stosowanie *isolated storage*:
- zmniejsza prawdopodobieństwo wystąpienia konfliktu między aplikacjami w zakresie ścieżek do plików;
- sprawia, że aplikacja staje się bardziej odporna na niekorzystny wpływ innych aplikacji na jej działanie;
- zwiększa przenośność aplikacji między różnymi wersjami i konfiguracjami Microsoft Windows.

Faktyczne miejsce przechowywania *isolated storage* na dysku jest **różne dla różnych wersji systemu operacyjnego Windows**.
Inne są również ścieżki do magazynów „przemieszczających się” (***roaming-enabled stores***) oraz magazynów „nie przemieszczających się” między stacjami klienckimi (***non-roaming stores***) w ramach tej samej wersji Microsoft Windows.

## Isolated Storage (4)

Przykłady ścieżek w systemach plików, gdzie przechowywane są izolowane magazyny na wybranych wersjach systemu Microsoft Windows:

- Windows 2000 (instalacja „od zera” – nie aktualizacja Windows NT 4.0), Windows XP
  *roaming-enabled stores*:
  `%SystemDrive%\Documents and Settings\<user>\Application Data`
  *non-roaming stores*:
  `%SystemDrive%\Documents and Settings\<user>\Local Settings\Application Data`
- Windows 2000 (aktualizacja Windows 4.0)
  *roaming-enabled stores*:
  `%SystemRoot%\Profiles\<user>\Application Data`
  *non-roaming stores*:
  `%SystemRoot%\Profiles\<user>\Local Settings\Application Data`

W *isolated storage* granica pomiędzy poszczególnymi magazynami (*stores*) jest wyznaczana w oparciu o tożsamość (*identity*): (1) użytkownika, (2) *assembly*, (3) domeny.



## Isolated Storage (5)

Użytkownicy są identyfikowani na podstawie tych samych właściwości, co w przypadku systemu operacyjnego.
Z każdym procesem uruchomionym w systemie operacyjnym Windows jest związana tożsamość użytkownika – oczywiście *impersonation* umożliwia zmianę żetonu związanego z dowolnym wątkiem aplikacji w trakcie działania.

O **tożsamości *assembly*** decydują następujące informacje zapisane w meta-danych *assembly*:

1. Tożsamość wydawcy (*publisher*) danego *assembly* zapisana w podpisie złożonym za pomocą Microsoft Authenticode®;
2. *Strong name* utworzone na zasadzie podpisania *assembly* kluczem prywatnym odpowiadającym zapisanemu w *assembly* kluczowi publicznemu;
3. Miejsce pochodzenia *assembly* (*origin*) zapisane w postaci URL.

Tożsamość *assembly* jest ustalana w oparciu o jedną z wymienionych cech, które są sprawdzane w przedstawionej wyżej kolejności (począwszy od najmocniejszej).

Przykładowo – jeśli w meta-danych *assembly* znajduje się zarówno podpis zgodny z formatem Microsoft Authenticode®, jak i *strong name* – w celach identyfikacji będzie wykorzystywana wyłącznie pierwsza informacja.

## Isolated Storage (6)

**Tożsamość domeny aplikacji (*domain identity*)** w kontekście *isolated storage* stanowi ścieżkę, z której uruchomiono dane *assembly*.
Dla aplikacji uruchomionej z internetu *domain identity* będzie URL miejsca, w którym umieszczono *assembly*, dla aplikacji uruchomionej z lokalnego komputera z kolei będzie to ścieżka w lokalnym systemie plików.

Microsoft .NET Framework dopuszcza następujące kombinacje cech, w oparciu o które wyznaczane są granice pomiędzy *isolated storages*:
- **tożsamość użytkownika oraz *assembly***;
- izolację w oparciu o **tożsamość użytkownika, *assembly* oraz domeny**.

Pierwszy z wymienionych rodzajów izolacji gwarantuje, że izolowane magazyny utworzone przez *assembly* A działające w kontekście bezpieczeństwa użytkownika U są dostępne wyłącznie dla tego *assembly* uruchomionego przez tego konkretnego użytkownika.

Izolacja w oparciu wyłącznie o tożsamość użytkownika i *assembly* umożliwia dostęp do danego *isolated store* w różnych aplikacjach, gdy są one uruchomione w kontekście tego samego użytkownika, oraz korzystają z tego samego *assembly*.
Z tego względu ten rodzaj izolacji powinien być stosowany w momencie, kiedy **istnieje potrzeba współdzielenia danych między różnymi aplikacjami**.

123

## Isolated Storage (7)

Domyślna polityka bezpieczeństwa dopuszcza tworzenie *isolated storage* w oparciu o połączenie tożsamości użytkownika oraz *assembly* dla *assemblies* znajdujących się w strefie: intranet.
Ze względu na możliwość przeciekania informacji między aplikacjami wymieniony sposób izolacji nie jest domyślnie dopuszczony dla aplikacji uruchamianych ze strefy: Internet.

Drugim rodzajem kryterium izolacji jest połączenie wszystkich cech: tożsamości użytkownika, *assembly* oraz domeny.

Jest to domyślny rodzaj izolacji dla *assemblies* uruchamianych ze strefy Internet, ponieważ uniemożliwia wyciekanie informacji poprzez *isolated storage*.

Wyznaczanie granic dla *isolated storage* w oparciu o trzy cechy uruchomionego *assembly* jest najbezpieczniejszym sposobem izolacji.
Z powyższego względu ten rodzaj izolacji powinien być stosowany wszędzie tam, gdzie chcemy skorzystać z *isolated storage*, a jednocześnie mamy wątpliwości dotyczące zasad izolacji.

124

# Isolated Storage (8)

Istnieje również możliwość skonfigurowania *isolated storage* tak, by podążało (*roam*) ono za użytkownikiem.
W przypadku „przemieszczającego się" izolowanego magazynu **nie ma możliwości wymuszenia ograniczeń dotyczących limitu użycia przestrzeni dyskowej (*quotas*)**.

Polityka bezpieczeństwa .NET Framework umożliwia administratorowi określenie:
- rodzaju *isolated storage*, który dane *assembly* może tworzyć, a następnie wykorzystywać w trakcie wykonywania;
- maksymalnego rozmiaru przestrzeni dyskowej, który dane *assembly* może wykorzystać dla swojego *isolated storage*.

Wymienione informacje są zapisane w – przypisanym do odpowiedniego *code group* – obiekcie **`IsolatedStorageFilePermission`**.

W celu przeglądania informacji nt. izolowanych magazynów utworzonych w systemie, całkowitego usuwania danych użytkownika z jego *isolated storage*, itp. administrator może posłużyć się specjalnie w tym celu dostarczonym narzędziem Isolated Store Admin: **`Storeadm.exe`**.



# Isolated Storage (9)

Wszystkie klasy związane bezpośrednio z izolowanym przechowywaniem plików zostały w .NET Framework zgrupowane w przestrzeni nazw: **`System.IO.IsolatedStorage`**.

Od strony programistycznej izolowane przechowywanie plików i katalogów wykorzystywanych przez *assembly* wymaga w pierwszej kolejności utworzenia izolowanego magazynu (*store*).
Magazyn jest reprezentowany przez zrealizowaną **w oparciu o wzorzec projektowy *factory*** klasę **`IsolatedStorageFile`**.

```
/// a store isolated by user and assembly identity
IsolatedStorageFile store1 =
   IsolatedStorageFile.GetUserStoreForAssembly();

/// equivalent of the above using a generic method GetStore,
/// which requires passing additional arguments defining the
/// type of isolated store with the IsolatedStorageScope
/// enumeration type
IsolatedStorageFile store2 = IsolatedStorageFile.GetStore(
   IsolatedStorageScope.User | IsolatedStorageScope.Assembly,
   null, null);
```

## Isolated Storage (10)

```
/// a store isolated by user, assembly and domain identity
IsolatedStorageFile store3 =
   IsolatedStorageFile.GetUserStoreForDomain();

/// equivalent of the above using a generic method GetStore
IsolatedStorageFile store4 = IsolatedStorageFile.GetStore(
   IsolatedStorageScope.User | IsolatedStorageScope.Assembly
   | IsolatedStorageScope.Domain, null, null);

/// creation of a roaming-enabled store requires usage of the
/// generic method GetStore
IsolatedStorageFile store5 = IsolatedStorageFile.GetStore(
   IsolatedStorageScope.User | IsolatedStorageScope.Assembly
   | IsolatedStorageScope.Roaming, null, null);
```

Izolowany magazyn powinien być zamknięty po zakończeniu korzystania z przechowywanych w nim plików i katalogów.

```
/// closing of a previously opened isolated store
isolatedStore.Close();
```



## Isolated Storage (11)

Dostęp do plików przechowywanych w *isolated storage* jest uzyskiwany za pomocą klasy **`IsolatedStorageFileStream`** rozszerzającej **`System.IO.FileStream`**.

Zazwyczaj w trakcie tworzenia instancji klasy **`IsolatedStorageFileStream`** należy podać izolowany magazyn, w którym dany plik będzie przechowywany. Istnieje jednak możliwość pominięcia *isolated store* w konstruktorze, co oznacza utworzenie pliku, którego **czas życia będzie odpowiadał czasowi istnienia danego wystąpienia klasy** **`IsolatedStorageFileStream`**.

```
/// creation of a file: "file.txt" for writing in the root
/// folder of the isolated store represented by isolatedStore
/// variable
IsolatedStorageFileStream stream =
   new IsolatedStorageFileStream( "file.txt", FileMode.Create,
      FileAccess.Write, isolatedStore );
/// opening a file: "file.txt" in the root folder of the
/// isolated store
IsolatedStorageFileStream stream =
   new IsolatedStorageFileStream( "file.txt", FileMode.Open,
      FileAccess.Read, isolatedStore );
```

# Isolated Storage (12)

Plik umieszczony w *isolated store* – podobnie jak każdy inny plik – należy zamknąć w momencie zakończenia korzystania, co oczywiście wiąże się z zapisem na dysk danych znajdujących się w buforze strumienia.

W celu odczytu lub zapisu do pliku w *isolated storage* można skorzystać z metod dostarczanych przez instancję klasy **`IsolatedStorageFileStream`**.
Oczywiście istnieje również możliwość utworzenia w oparciu o tę instancję obiektów klas wykorzystywanych standardowo w .NET Framework do odczytu i zapisu do strumienia:

- **`StreamReader`**, **`StreamWriter`** – dla strumieni tekstowych, oraz
- **`BinaryReader`**, **`BinaryWriter`** – dla strumieni binarnych.

```
/// method storing a string in a file located in isolated store
void WriteStream (IsolatedStorageFile st, string f, string s) {
   IsolatedStorageFileStream s =
      new IsolatedStorageFileStream( f, FileMode.Create,
         FileAccess.Write, st );
   using (StreamWriter sw = new StreamWriter(s)) {
      sw.Write(s); sw.Close(); /// writer closes the stream
   } /// file handle is released
}
```



# Isolated Storage (13)

*Isolated storage* oczywiście udostępnia inne operacje wspierane zazwyczaj przez system plików jak np. tworzenie katalogów.

```
IsolatedStorageFile isoStore =
   IsolatedStorageFile.GetUserStoreForDomain();
/// creation of a directory in the root folder of isolated
/// store
isoStore.CreateDirectory("Directory");
/// creation of a subdirectory of the previously created
/// directory
isoStore.CreateDirectory("Directory/Subdirectory");
```

W rzeczywistości druga z powyższych instrukcji tworzy dwa katalogi, przy czym wyjątek nie jest zgłaszany jeśli tworzony katalog już istnieje – jak ma to miejsce w przedstawionym przypadku katalogu: **`"Directory"`**.

Wyjątek jest natomiast zgłaszany, jeśli w nazwie katalogu występują niedozwolone znaki, np. symbole wieloznaczne (*wildcards*): **`"*"`** lub **`"?"`**.

## Isolated Storage (14)

Oczywiście możliwości tworzenia plików i katalogów w *isolated storage* odpowiada funkcjonalność pozwalająca na ich usuwanie.

```
IsolatedStorageFile isoStore =
   IsolatedStorageFile.GetUserStoreForDomain();
isoStore.CreateDirectory("Directory");
/// deletion of a previously created directory
isoStore.DeleteDirectory("Directory");
string fileName = "file.txt";
IsolatedStorageFileStream stream =
   new IsolateStorageFileStream(fileName, FileMode.Create,
      FileAccess.Write, isoStore);
/// deletion of a file placed in the isolated store
isoStore.DeleteFile(fileName);
```

Próba usunięcia nieistniejącego pliku bądź katalogu powoduje zgłoszenie wyjątku. Jako argument nie można również przekazywać nazwy zawierającej symbole wieloznaczne (*wildcards*).

Nie jest również dopuszczalne usuwanie katalogu zawierającego pliki bądź podkatalogi – w takim wypadku należy w pierwszej kolejności usunąć całą zawartość usuwanego katalogu.

131

## Isolated Storage (15)

Podczas wydobywania zawartości *isolated store* (plików i folderów) można korzystać z *wildcards*.

```
IsolatedStorageFile isoStore =
   IsolatedStorageFile.GetUserStoreForDomain();
/// get the list of all the directories placed in the root
/// folder of the isolated store
string[] subdirs = isoStore.GetDirectoryNames("*");
/// analogously get the list of all the files located in the
/// root folder of the isolated storage
string[] files = isoStore.GetFileNames("*");
```

Rozmiar izolowanego magazynu może być ograniczony limitami (*quotas*) narzuconymi przez politykę bezpieczeństwa .NET Framework.
Dopuszczalny rozmiar *isolated storage* może być odczytany z właściwości: **`MaximumSize`**.

```
IsolatedStorageFile isoStore =
   IsolatedStorageFile.GetUserStoreForDomain();
/// establish the still available space in the isolated storage
ulong spaceLeft = isoStore.MaximumSize – isoStore.CurrentSize;
```

132

## Authorization Manager (1)

**Discretionary Access Control Lists (DACL)** stanowią podstawę kontroli dostępu do zasobów na platformie Microsoft Windows od czasu wprowadzenia *Private Object Security API* w Windows NT 4.0.

W modelu ACL przyjętym w Windows użytkownik może ograniczyć dostęp do zasobu poprzez dołączenie do niego DACL.
Kontrola dostępu polega na porównaniu zawartości ACL danego obiektu z tożsamością użytkownika oraz grup, do których ten użytkownik należy.

Poważną wadą tego rodzaju autoryzacji jest fakt, że aplikacje, w których trudno jest w sposób naturalny zdefiniować trwałe obiekty bardzo trudno poddają się kontroli dostępu w oparciu o ACL.

Model oparty o ACL wymusza na administratorze tłumaczenie polityki autoryzacji (bezpieczeństwa) istniejącej w organizacji na występujące na niskim poziomie uprawnienia do obiektów.



## Authorization Manager (2)

W przypadku aplikacji biznesowych (*line-of-business*) bardzo trudno – zwłaszcza na etapie tworzenia aplikacji – jest przełożyć prawa dostępu na uprawnienia do określonych trwałych obiektów.

O wiele bardziej stosowny jest tutaj przydział uprawnień, który:
- wynika z przebiegu procesów pracy (*workflow*) – czynności wykonywane w ramach procesu biznesowego są wykonywane przez określoną rolę w procesie;
- umożliwia realizację określonych operacji, jak np. wysłanie zapytania do bazy danych, czy wysłanie wiadomości pocztą elektroniczną.

W aplikacjach biznesowych decyzja o przyznaniu, bądź odmowie dostępu jest ściśle związana z regułami panującymi w logice biznesu wspieranej przez tę aplikację – jak np. wielkość wydatków, bądź weryfikacja zakończenia procesu pracy (przykładowo: zatwierdzenie *deliverables* projektu przez przełożonego)

Realizacja własnego (*proprietary*) mechanizmu autoryzacji koncepcyjnie związanego ze strukturą organizacyjną:
- podnosi koszty wytworzenia rozwiązania, a także
- zwiększa złożoność zadań administracyjnych ze względu na rozbieżne modele autoryzacyjne istniejące w aplikacji i platformie, na której jest ona uruchomiona.

## Authorization Manager (3)

**Authorization Manager (AzMan)** wprowadza nowy rodzaj autoryzacji na platformie Windows.
Jest to **prosty w użyciu szkielet (*authorization framework*)** dostosowany w większym stopniu niż DACL do potrzeb aplikacji biznesowych, w których istnieje potrzeba przyznawania uprawnień zgodnie funkcją pełnioną przez użytkownika w organizacji.

Authorization Manager wspiera tzw. **kontrolę dostępu opartą o role (Role-Based Access Control – RBAC)**, która umożliwia administratorowi aplikacji określenie kontroli dostępu w kontekście struktury organizacyjnej instytucji.

RBAC określa uprawnienia w kategoriach użytkowników oraz ról (*job roles*) – funkcji pełnionych przez użytkowników w ramach organizacji.

W modelu RBAC funkcje pełnione w ramach organizacji są mapowane na uprawnienia w aplikacji.

Tym samym RBAC upraszcza administrację zapewniając elastyczniejsze oraz bardziej naturalne zarządzanie bezpieczeństwem w środowiskach korporacyjnych.

135

## Authorization Manager (4)

Przykładowo w aplikacji wspierającej sprzedaż może występować rola odpowiadająca stanowisku kierownika sprzedaży.

Rola kierownika sprzedaży posiada uprawnienia wystarczające osobie pracującej na odpowiadającym jej stanowisku pracy na korzystanie z systemu informatycznego w zakresie wystarczającym do wypełniania jej codziennych obowiązków.

W modelu RBAC administrator wykorzystuje rolę w celu zarządzania uprawnieniami oraz przypisaniem uprawnień do użytkowników.

W momencie zatrudnienia konkretnej osoby na stanowisko kierownika sprzedaży jej konto użytkownika zostaje przypisane do danej roli.
Opuszczeniu tego stanowiska przez wspomnianą osobę będzie towarzyszyła likwidacja skojarzenia jej konta z daną rolą.

Ponieważ RBAC umożliwia przyznawanie uprawnień w kontekście modelu organizacyjnego instytucji, przypisywanie uprawnień jest stanowczo bardziej naturalne i intuicyjne dla administratora aplikacji.

136

## Authorization Manager (5)

W pewnym zakresie RBAC może być zastąpione przez grupy użytkowników – w takim wypadku grupa odpowiada roli użytkownika (pracownika) w organizacji. Administratorzy aplikacji mogą określić uprawnienia wymagane dla roli poprzez przydzielenie uprawnień odpowiedniej grupie w DACL danego obiektu.

Niestety wraz ze wzrostem liczby obiektów (do których wymagany jest dostęp) rośnie także liczba miejsc, w których administrator musi wprowadzać (w sposób spójny) zmiany w momencie modyfikacji uprawnień konkretnego użytkownika.

Konsekwentne korzystanie z grup zasobów (np. katalogów) oraz grup użytkowników może ograniczyć wysiłek z tym związany, ale wymaga spójnej, precyzyjnie określonej i bezwzględnie przestrzeganej praktyki definiowania grup zasobów, a także koordynacji prac wykonywanych przez administratorów.

Jednym z elementów wykonywanego okresowo audytu bezpieczeństwa jest określenie uprawnień poszczególnych użytkowników.
W przypadku ACL określenie praw dostępu użytkownika, bądź grupy użytkowników wiąże się z odpytaniem ACL dołączonego do każdego obiektu.
Jest to zatem tym trudniejsze, im większa jest liczba obiektów chronionych za pomocą ACL.

137

## Authorization Manager (6)

Oczywiście dziedziczenie uprawnień z katalogów może ten proces usprawnić, ale niestety bez gwarancji, że ACL przypisane poszczególnym obiektom są zgodne z instytucjonalną polityką bezpieczeństwa.

Precyzyjne ustalenie bieżącej polityki autoryzacji wymaga odpytania ACL związanych z wieloma obiektów, co sprawia, że uprawnienia użytkowników i grup użytkowników są raczej rzadko weryfikowane.

W odróżnieniu do grupy użytkowników w katalogu Windows rola w RBAC określa uprawnienia do zbioru zasobów – z tego względu określenie uprawnień poszczególnych użytkowników nie wymaga przeglądania wszystkich zasobów.

W większości środowisk raz ustalone uprawnienia roli będą podlegać rzadkiej modyfikacji.
Zmianie może natomiast ulegać przypisanie użytkowników do roli.

Głównym zadaniem administratorów po ustaleniu ról będzie zatem zarządzanie przynależnością do poszczególnych ról – nie uprawnieniami do konkretnych obiektów, jak miało to miejsce w przypadku ACL.

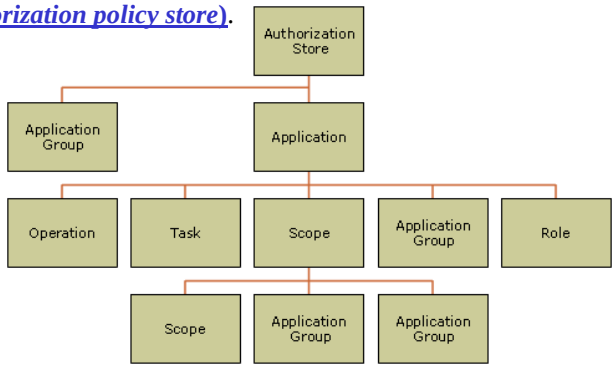
138

# Authorization Manager (7)

Polityka autoryzacji jest zazwyczaj rozproszona – przechowywana razem z zasobami, do których dostęp ogranicza – w NTFS ACL towarzyszy plikowi, którego dotyczy.

Kojarzenie uprawnień z grupami zasobów sprawia, że polityka autoryzacji w RBAC jest mniej rozproszona niż zazwyczaj.
W Authorization Manager polityka autoryzacji nie jest – w odróżnieniu do ACL – bezpośrednio powiązana z zasobami i w całości znajduje się w tzw. **magazynie polityki autoryzacji (*authorization policy store*)**.

Elementy wchodzące w skład *authorization store* oraz występujące pomiędzy nimi zależności:






# Authorization Manager (8)

**Operacja** stanowi uprawnienie niskiego poziomu reprezentujące uprzywilejowaną akcję, bądź fragment funkcjonalności aplikacji.
Zazwyczaj pojedyncza operacja nie wystarcza dla realizacji zadania wyższego poziomu – rozpoznawalnego z perspektywy logiki biznesu wspieranej przez aplikację.

Z punktu widzenia aplikacji operacja jest zawsze wykonywana **jako niepodzielna całość i jako jedna całość jest zabezpieczana**.

Nazwy operacji są najczęściej zrozumiałe wyłącznie dla twórców rozwiązania i – zazwyczaj – nic nie mówią osobom odpowiedzialnym za administrację aplikacji.
Z tego względu operacje raczej nie powinny być w ogóle opisywane w dokumentacji użytkowej.

W modelu RBAC pojedyncza operacja może być **wykorzystywana przez wiele zadań wyższego poziomu**.

## Authorization Manager (9)

Przykładowo operacja odczytująca zgromadzone w bazie danych informacje nt. zamówienia sama w sobie nie stanowi funkcjonalności zadania zrozumiałego na poziomie abstrakcji dziedziny problemowej.

Wymieniona operacja może natomiast wchodzić w skład dwóch różnych zadań: (1) obsługi zamówienia, oraz (2) sprawdzenia statusu zamówienia.

W przypadku idealnym operacja powinna reprezentować każdą funkcję aplikacji wymagającą kontroli dostępu: każdą metodę, bądź polecenie przekazywane do bazy danych.

Im mniejsza granularność operacji, tym większa elastyczność w trakcie określania i przyznawania uprawnień.

Należy jednak pamiętać, że zbyt mała granularność operacji niepotrzebnie zwiększa złożoność administracji aplikacji.

141

## Authorization Manager (10)

Najczęściej poziom abstrakcji operacji jest zbyt niski dla administratora aplikacji. W celu utworzenia uprawnień mających znaczenie dla administratora programista powinien pogrupować operacje w zadania.

**Zadanie** jest zestawem operacji niskiego poziomu, którego celem jest określenie, które operacje są wymagane w trakcie **realizacji określonej funkcjonalności aplikacji zrozumiałej z perspektywy administratora aplikacji**.

Zadania usprawniają zarządzanie rolami oraz umieszczają uprawnienia do poszczególnych funkcji aplikacji w kontekście czynności zrozumiałych w ramach wspieranej logiki biznesu – czynności podejmowanych w trakcie realizacji procesów biznesowych istniejących w organizacji.

Przykładami zadań są: umieszczenie dokumentu w systemie zarządzania treścią, wpisanie terminu spotkania w wykorzystywanym w przedsiębiorstwie systemie pracy grupowe, złożenie podpisu elektronicznego, zmiana hasła użytkownika, itp.

Oczywiście przed przystąpieniem do definiowania zadania w *policy store* należy w sposób precyzyjny ustalić, które z operacji niższego poziomu są konieczne w trakcie realizacji zadania rozpoznawalnego w dziedzinie problemowej.

142

## Authorization Manager (11)

Zadania – poza atomowymi operacjami – mogą grupować inne zadania.

Przykładowo zadanie polegające na zarządzaniu kontem użytkownika może obejmować kilka pomniejszych zadań: (1) zmiana hasła użytkownika, (2) wyzerowanie hasła użytkownika oraz (3) dezaktywuj konto użytkownika, itp.
Z kolei wyrażenie zgody na publikację artykułu może wymagać złożenia podpisu elektronicznego przez redaktora odpowiedzialnego za treść umieszczaną na portalu.

Obiekty reprezentujące zadania w ramach Authorization Manager posiadają dodatkową funkcjonalność polegającą na dynamicznym ustalaniu uprawnień przydzielanych w trakcie wykonywania zadania w oparciu o tzw. **reguły biznesowe**.

**Reguły biznesowe (*business rules – BizRules*)** lub też ***authorization scripts*** to skrypty w VBScript, bądź JScript, które są dołączane do obiektów reprezentujących zadania.

Reguły biznesowe są uruchamiane w momencie zgłoszenia przez użytkownika żądania dostępu do zasobu.



## Authorization Manager (12)

Fakt wywoływania *BizRules* w chwili pojawienia się żądania od użytkownika sprawia, że w trakcie podejmowania decyzji dotyczącej przyznania dostępu do zasobu istnieje możliwość **uwzględnienia informacji dostępnej wyłącznie w trakcie wykonywania aplikacji** – np. bieżącą godzinę, lub kwotę transakcji.

Jeśli reguła biznesowa zakończy się powodzeniem aplikacja działająca w imieniu użytkownika może wykonać operacje wchodzące w skład zadania.
Jeśli *BizRule* się nie powiedzie użytkownik nie może uruchomić żadnej operacji związanej z zadaniem.

Oczywiście nie znaczy to, że użytkownik nie może mieć możliwości wykonywania tego samego zestawu operacji za pośrednictwem innego zadania.

*BizRules* są wyzwalane w momencie wywołania metody **`AccessCheck`**, za pośrednictwem której programista aplikacji może przekazać parametry do reguły biznesowej.

Parametry są wysyłane w postaci dwóch tablic przechowujących odpowiadające sobie nazwy i wartości parametrów.
Parametry i odpowiadające im wartości **muszą być przechowywane pod tymi samymi indeksami** wspomnianych tablic i być przekazane **zgodnie z kolejnością alfabetyczną nazw parametrów**.

## Authorization Manager (13)

Reguły biznesowe mogą być dodane przez administratora już po wdrożeniu aplikacji.

Oznacza to, że w trakcie tworzenia aplikacji programista, bądź projektant aplikacji mogą nie wiedzieć, czy podczas wywoływania **`AccessCheck`** w danym fragmencie kodu powinny być przekazane parametry, które mogłyby być w przyszłości skonsumowane przez *BizRule*.

Z tego względu aplikacja musi przekazywać parametry, które **mogą być potencjalnie przydatne dla reguły biznesowej**.

Informacja, która powinna być przekazywana przez **`AccessCheck`** do reguł biznesowych obejmuje nazwę użytkownika oraz dane, które mogą być wykorzystywane w trakcie określania ograniczeń dostępu.

Dla ułatwienia można zdefiniować zestaw parametrów, który jest przekazywany w trakcie każdego wywołania **`AccessCheck`**.

Typowym przykładem parametru, który może być przydatne w trakcie ewaluacji wywołania **`AccessCheck`** umieszczonego w dowolnym kontekście jest – poza nazwą użytkownika – bieżąca data.

145

## Authorization Manager (14)

Lista parametrów przekazywanych do każdego wywołania powinna być uzupełniana parametrami, które mogą występować tylko w kontekście wykonywania określonego zadania.

Przykładem tego rodzaju danych mogą być: koszt zamówienia składanego przez użytkownika, bądź informacja o jego kierowniku – choć ta ostatnia powinna być możliwa do uzyskania pośrednio za pomocą nazwy użytkownika, itp.

Nie można zapominać, że tworzenie *BizRules* wymaga pewnej wiedzy w zakresie pisania skryptów, która to umiejętność nie jest dana większości administratorów aplikacji.

Z tego względu optymalnym rozwiązaniem byłoby, gdyby reguły biznesowe zostały dostarczone razem z aplikacją, bądź były stworzone w późniejszym terminie przez osoby zajmujące się programowaniem – nie administracją.

Reguły biznesowe powinny być **procedurami o bardzo niewielkiej złożoności przetwarzania** polegającymi np. na porównaniu wartości parametrów, bądź wykonaniu zapytania do katalogu użytkowników.

146

## Authorization Manager (15)

Wyniki działania reguł biznesowych są – w celu poprawienia wydajności ewaluacji często wywoływanych *BizRule* – przechowywane (*cached*) w kontekście użytkownika.

```
// example business rule ensuring that the time of day is
// between 9:00 and 17:00
AzBizRuleContext.BusinessRuleResult = false;
dt = new Date();
hour = dt.getHours();
if (hour > 9 && hour < 17) {
   AzBizRuleContext.BusinessRuleResult = true;
}
```

Jeśli reguła biznesowa korzysta z parametrów przekazanych jej przez aplikację, metoda **`AccessCheck`** przechowuje wynik wywołania BizRule **wiążąc wynik reguły z wartościami parametrów przekazanych do reguły**.

Tym samym podnoszona jest wydajność podejmowania decyzji dotyczącej przyznania dostępu do wykonania zadania – oczywiście, o ile nie zaszły zmiany w kontekście wywoływania **`AccessCheck`**.



## Authorization Manager (16)

```
// examplary business rule ensuring that the amount passed
// is less than 500
AzBizRuleContext.BusinessRuleResult = false;
amount = AzBizRuleContext.GetParameter("ExpenseAmount");
if (amount < 500) {
   AzBizRuleContext.BusinessRuleResult = true;
}
```

Wynik reguł biznesowych jest często zależny nie tylko od przekazywanych do nich wartości poszczególnych parametrów przekazywanych, ale np. od czasu – jak w przykładzie BizRule ograniczającej dostęp do godzin pomiędzy 9:00 a 17:00.

Wynik wykonania reguły biznesowej przechowywany w kontekście użytkownika po pewnym czasie może być już nieaktualny, co pociągnie za sobą **niewłaściwą decyzję w zakresie kontroli dostępu**.

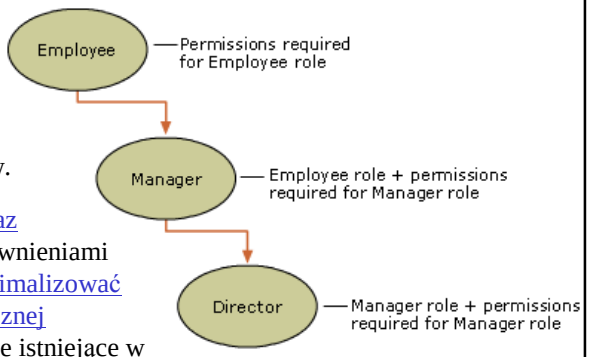
Z tego względu kontekst użytkownika powinien być w stosunkowo niewielkich odstępach czasu czyszczony przez aplikację.

# Authorization Manager (17)

**Stanowiąca fundament RBAC rola** odpowiada zestawowi uprawnień, które muszą posiadać użytkownicy, tak by mogli wykonywać określone czynności.
Rola powinna odpowiadać **rzeczywistej funkcji** istniejącej w ramach organizacji.

**Uprawnienia** są w RBAC **definiowane jako zestaw zadań i operacji**.
Użytkownicy są przypisywani do roli, z kolei rola jest przypisywana do zbioru zasobów.

Ponieważ tworzenie ról wpływa bezpośrednio na uprawnienia użytkowników aplikacji, jest to proces wymagający szczególnej uwagi ze strony administratorów.




Problemy związane z analizą oraz późniejszym zarządzaniem uprawnieniami poszczególnych ról można zminimalizować poprzez zastosowanie hierarchicznej zależności ról wykorzystując role istniejące w trakcie definicji nowych ról.



# Authorization Manager (18)

**Zakres (*scope*)** – zestaw obiektów lub zasobów, których dotyczy odrębna polityka autoryzacji.

Zakres może reprezentować zbiór konkretnych fizycznych obiektów, np. katalog, lub elastyczną regułę umożliwiającą wyszukiwanie obiektów zapisaną przy użyciu symboli wieloznacznych (*wildcards*) np. *.txt.

Aplikacje wykorzystują zakresy w celu grupowania zasobów, zatem muszą być w stanie zmapować żądane zasoby do określonego zakresu w trakcie sprawdzania uprawnień podczas wykonywania.

W ramach aplikacji – poza jawnie zdefiniowanymi zakresami – **istnieje tzw. zakres domyślny (*default scope*)**.

Uprawnienia nadane rolom zdefiniowanym wewnątrz *default scope* pozostają w mocy również w pozostałych zakresach zdefiniowanych w ramach obiektu reprezentującego aplikację.

## Authorization Manager (19)

**Grupa użytkowników aplikacji (*application group*)** – lista użytkowników lub/i grup użytkowników przechowywanych: (1) globalnie w *authorization store*, (2) w aplikacji zdefiniowanej wewnątrz *authorization store*, bądź (3) zakresu w ramach aplikacji.

Miejsce przechowywania grupy aplikacyjnej określa obszar, którego dana grupa dotyczy: (1) całość *authorization store*, (2) pojedyncza aplikacja, bądź (3) zakres w ramach wybranej aplikacji.

Authorization Manager udostępnia dwa rodzaje grup aplikacyjnych:
- podstawowe grupy aplikacyjne (*application basic groups*), oraz
- grupy aplikacyjne powstałe w wyniku zapytania LDAP (*LDAP-query groups*).

**Podstawowa grupa aplikacyjna** – lista użytkowników, bądź grup użytkowników z katalogu użytkowników Windows, bądź innych grup aplikacyjnych.

W odróżnieniu do grupy użytkowników Windows definicja podstawowej grupy aplikacyjnej może również określać użytkowników, którzy nie wchodzą w skład danej grupy aplikacyjnej (*non-members*).
Tym samym możliwe jest wyłączanie mniejszej grupy, bądź pojedynczych użytkowników z większej grupy użytkowników.

151

## Authorization Manager (20)

Podstawowe grupy aplikacyjne przechowują identyfikatory bezpieczeństwa (*security identifiers – SIDs*) każdego użytkownika bądź grupy użytkowników należących do grupy aplikacyjnej (*members*), bądź wykluczonych z tej grupy (*non-members*).

Z tego względu w miarę zwiększania zawartości listy członków grupy aplikacyjnej zwiększa się ilość informacji przechowywanej wewnątrz *authorization store*, co w konsekwencji wydłuża czas inicjalizacji magazynu autoryzacji, aplikacji, bądź zakresu.

**Grupa aplikacyjna LDAP (*LDAP-query group*)** – lista użytkowników aplikacji powstała w wyniku zapytania o wartości atrybutów związanych z kontem użytkownika w protokole LDAP wysłanego do Active Directory.

Grupy użytkowników przechowywane wewnątrz Active Directory zazwyczaj odpowiadają pewnym zbiorowościom istniejących w ramach dziedziny problemowej wspieranej przez system informatyczny.

Przykładowo pracownicy działu sprzedaży są w ramach Active Directory zgrupowani wewnątrz jednej grupy użytkowników, z kolei pracownicy działu marketingu są członkami innej grupy użytkowników.

152

## Authorization Manager (21)

Informacja o przynależności do danej grupy użytkowników jest jednym z atrybutów obiektu reprezentującego konto użytkownika w ramach Active Directory.

W momencie przeniesienia pracownika do innego działu administrator aplikacji musi dokonać spójnej zmiany zarówno w Active Directory, jak i polityce autoryzacji samej aplikacji.

Wykorzystanie zapytania LDAP w momencie określania uprawnień użytkownika umożliwia zdefiniowanie elastycznych reguł przynależności do grupy aplikacyjnej.

Zapytania LDAP poprawiają bezpieczeństwo aplikacji poprzez uproszczenie czynności administracyjnych polegającym na zwolnieniu administratora z obowiązku dokonywania spójnych modyfikacji polityki autoryzacji w wielu miejscach.

```
// returns all members of the group: Students of Information
// Systems major
(memberOf= CN=Students,OU=Information Systems Department,
   DC=pjwstk,DC=edu,DC=pl)
```

153

## Authorization Manager (22)

```
// returns all users, which are at least 18 years old
(age>=18)
```

```
// returns all users located in Poland
(country=Poland)
```

```
// returns all users located in Poland and over 18 years old
(&(country=Poland)(age>=18))
```

Kiedy aplikacja korzystająca z Authorization Manager jest inicjalizowana, ładuje politykę autoryzacji z określonego **magazynu polityki autoryzacji (*authorization store*)**.

Authorization Manager umożliwia zapis *authorization store* w Active Directory, bądź w postaci dokumentu .XML o ściśle określonym schemacie.

Korzystanie z grup aplikacyjnych LDAP wymaga by, konta użytkowników były umieszczone w Active Directory, ale nic nie stoi na przeszkodzie, by sama polityka autoryzacji nie mogła być przechowywana w pliku XML.

154

## Authorization Manager (23)

Active Directory jest odpytywane za pośrednictwem LDAP w momencie wywołania metody **`AccessCheck`**, co oczywiście nie pozostaje bez wpływu na czas realizacji wywołania tej metody.

Wyniki zapytań do Active Directory mogą być – podobnie jak wyniki wykonania *BizRules* – przechowywane w kontekście użytkownika – rzecz jasna ze wszystkimi tego konsekwencjami.

Korzystanie z Active Directory jako magazynu przechowującego politykę autoryzacji wymaga, by AD posiadało funkcjonalność dostępną w Windows Server 2003.

Zastosowanie odpowiedniej łaty na systemie Windows 2000 Server SP4 stwarza możliwość korzystania z funkcjonalności Authorization Manager w katalogach AD opartych również o tę wersję Windows.

Authorization Manager tworzy dla danego *authorization store* węzeł w drzewie AD, oraz podwęzły dla każdego elementu składającego się na definicję polityki autoryzacji: (1) grupy aplikacyjnej, (2) aplikacji, (3) operacji, (4) zadania, roli, czy (5) zakresu.

155

## Authorization Manager (24)

Korzystanie z Active Directory, jako miejsca przechowywania polityki autoryzacji umożliwia większy wpływ aplikacji na moment, w którym obiekty reprezentujące aplikacje będą załadowane do pamięci operacyjnej.

W momencie podłączenia aplikacji do *policy store* w przestrzeni adresowej aplikacji **tworzona jest lokalna kopia tymczasowa atrybutów (węzłów)** przechowywanych w *store* na poziomie globalnym:

- grup aplikacyjnych przechowywanych na poziomie globalnym, oraz
- nagłówków dla danych aplikacji zapisanych w *store*.

Kiedy aplikacja **inicjalizuje politykę autoryzacji dla danego obiektu aplikacji** poprzez wywołanie metody **`OpenApplication`**, do pamięci aplikacji ładowane są **wszystkie węzły-gałęzie znajdujące się w poddrzewie** reprezentującym daną aplikację.

Lokalne kopie tych danych są przechowywane dopóki obiekt reprezentujący aplikację nie zostanie usunięty.

Zdefiniowane w ramach danej aplikacji zakresy są również ładowane na żądanie – albo w wyniku wywołania metody **`AccessCheck`**, bądź jawnie poprzez metodę **`OpenScope`**.

156

## Authorization Manager (25)

Lokalna kopia tymczasowa polityki autoryzacji dotyczącej zakresu jest przechowywana **dopóki nie zostanie usunięty z pamięci cały obiekt reprezentujący aplikację**, wewnątrz której zdefiniowano dany zakres.

Active Directory nie wspiera zapisów transakcyjnych na wielu obiektach, bądź atrybutach.
*Authorization store* nie jest zatem odporne na uszkodzenie w wyniku równoczesnej modyfikacji – np. za pomocą interfejsu graficznego dostarczanego przez AzMan.

Ze względu na tworzenie tymczasowej lokalnej kopii polityki przechowywanej w kontrolerze domeny w trakcie inicjalizacji, **nie zaleca się stosowania grup aplikacyjnych dla grup zawierających powyżej 2000 użytkowników**.

W zamian **należy korzystać z grup użytkowników zdefiniowanych w Active Directory**, które nie są kopiowane do przestrzeni adresowej aplikacji.

Kopiowanie całej polityki autoryzacji sprawia również, że aplikacja korzystająca z AzMan wymagają szerokopasmowego dostępu do kontrolera domeny przechowującego tę politykę – nawet jeśli sama aplikacja może realizować swoją funkcjonalność w oparciu o łącza o niskiej przepustowości.

157

## Authorization Manager (26)

*Authorization store* przechowywane w Active Directory podlega replikacji na wszystkie kontrolery domeny, w której polityka autoryzacji jest przechowywana.

W celu zmniejszenia obciążenia sieci wynikającego z replikacji obiektów zapisanych w *authorization store* istnieje możliwość umieszczenia *policy store* w osobnej domenie.
Stworzenie osobnej domeny dla *policy store* **wymaga utworzenia relacji zaufania** między domeną przechowującą konta użytkowników, a domeną zawierającą politykę autoryzacji.

Authorization Manager wspiera również **zapis polityki autoryzacji w dokumencie XML**, w którym każdy z węzłów jest **umieszczony pod unikalnym identyfikatorem (*globally unique identifier – GUID*)**.

Ponieważ schemat dokumentu przechowującego authorization store nie został opublikowany przez Microsoft, nie należy modyfikować tego dokumentu ręcznie.

Pozostaje zatem możliwość modyfikacji polityki autoryzacji za pomocą:
- narzędzia Authorization Manager MMC, albo
- API udostępnianego przez Authorization Manager w językach skryptowych: VBScript oraz JScript.

158

## Authorization Manager (27)

Authorization Manager wykorzystuje model aplikacji oparty o **tzw. zaufany podsystem (*trusted subsystem application model*)**.

*Trusted subsystem application model* stanowi rozszerzenie dla znanego z Windows 2000 oraz Windows NT Server modelu opartego o „wcielanie się" (*impersonation*), w którym kontrola dostępu polegała na porównaniu ACL zasobu z żetonem towarzyszącym wątkowi procesu.

W modelu opartym o zaufany podsystem jedynie konto aplikacyjne posiada uprawnienia umożliwiające wykonywanie wszystkich operacji udostępnianych użytkownikom.

W momencie zażądania usługi przez użytkownika aplikacja realizująca logikę biznesową jest odpowiedzialna za przeprowadzanie kontroli dostępu w oparciu o politykę autoryzacji związaną z tą aplikacją.

Jeśli użytkownik posiada uprawnienia do określonego zadania – aplikacja wykonuje to zadanie w jego imieniu.

W modelu opartym o zaufany podsystem – w odróżnieniu do *impersonation* – **użytkownicy nie posiadają bezpośredniego dostępu do zasobów**.

159

## Authorization Manager (28)

W przypadku *impersonation* przypisanie DACL do zasobów w celu określenia uprawnień użytkownika w zakresie wykonywania zadań wysokiego poziomu może być dość kłopotliwe.

W praktyce administratorzy przyznają zatem większe uprawnienia, niż wynikają z polityki bezpieczeństwa – w celu oszczędzenia czasu.

W przypadku *trusted subsystem model* dostęp do zasobów jest wykonywany w kontekście bezpieczeństwa konta aplikacji, co – w porównaniu do *impersonation* – skutecznie upraszcza zawartość DACL.

W DACL zapisane są jedynie uprawnienienia, które musi posiadać usługa, tak by była w stanie zrealizować w imieniu użytkownika całą dostarczaną przez siebie funkcjonalność.

Należy pamiętać, by konto w kontekście bezpieczeństwa którego będzie działała usługa nie posiadało uprawnień administratora, a jedynie taki zestaw uprawnień, który umożliwi realizację całej funkcjonalności udostępnianej przez aplikację.

Zakres uprawnień wymaganych przez usługę powinien być precyzyjnie udokumentowany przez twórców aplikacji.

160

## Authorization Manager (29)

W *impersonation* użytkownik uzyskuje bezpośredni dostęp do zasobów, co stwarza niebezpieczeństwo, że na zasobie może zostać wykonana operacja inna, niż przewidywali twórcy aplikacji.

W przypadku zaufanego podsystemu jedynie konto aplikacyjne ma dostęp do zasobu, a sama aplikacja jest odpowiedzialna za autoryzację, zatem użytkownik nie może wykonać operacji spoza zbioru udostępnionego przez twórców aplikacji.

Wymaganie bezpośredniego dostępu do zasobów sprawia również kłopoty natury wydajnościowej – zazwyczaj bowiem konieczne jest tworzenie osobnego połączenia do bazy danych w celu obsługi pojedynczego żądania klienckiego.

Model oparty o zaufany podsystem opiera się na autoryzacji po stronie samej aplikacji.
Z tego względu jedno połączenie do bazy danych może być wykorzystywane w trakcie obługi żądań pochodzących od wielu użytkowników.

161

## Authorization Manager (30)

Od strony implementacyjnej **Authorization Manager jest obiektem COM** uruchomionym na stacji, na której została uruchomiona aplikacja.
Za pośrednictem zestawu interfejsów COM AzMan umożliwia programiście aplikacji proste zarządzanie *policy store* oraz weryfikację uprawnień użytkowników do żądanych zadań grupujących operacje realizowane przez aplikację.

Korzystanie z AzMan nie wymaga uruchomienia żadnych dodatkowych usług, bądź aplikacji na kontrolerze domeny.

Aplikacje korzystające z AzMan wykorzystują udostępniane interfejsy COM w celu walidacji żądań klienckich.

Aplikacja tworzona w .NET Framework może korzystać z AzMan za pośrednictem ***tzw. interop assembly***.

*Interop assembly* umożliwia komunikację między określonym obiektem COM zarejestrowanym i uruchomionym na komputerze a kodem uruchomionym pod kontrolą CLR.

*Interop assembly* służące do komunikacji z AzMan jest standardowo instalowany w katalogu: **`%WINDIR%\Microsoft.NET\AuthMan`**.

162

## Authorization Manager (31)

Pierwszą rzeczą, jaką należy wykonać w aplikacji korzystającej z AzMan jest inicjalizacja AzMan oraz określonego *authorization policy store*.

```
IAzAuthorizationStore store = new AzAuthorizationStoreClass();
store.Initialize(0,@"msxml://C:\AzStore.xml", null);
IAzApplication application = store.OpenApplication(
   "My Application", null);
```

W następnej kolejności – o ile wcześniej zostało przeprowadzone już uwierzytelnienie użytkownika – użytkownika należy utworzyć kontekst bezpieczeństwa użytkownika w Authorization Manager.

Jeśli uwierzytelnie wykorzystywane przez aplikację zostało oparte o mechanizmy dostarczane w tym zakresie przez Microsoft Windows istnieje możliwość stworzenia kontekstu użytkownika w oparciu o żeton związany z wątkiem aplikacji.

```
/// previous code snippet continued
WindowsIdentity id = (WindowsIdentity)User.Identity;
IntPtr token = id.Token;
IAzClientContext context
   = application.InitializeClientContextFromToken(token, null);
```



## Authorization Manager (32)

Tworzenie kontekstu bezpieczeństwa w oparciu o żeton nie wymaga w dalszej kolejności odpytywania kontrolera domeny w celu uzyskania informacji nt. grup, do których należy uwierzytelniony użytkownik.

W przypadku, gdy aplikacja nie korzysta z uwierzytelnienia Windows istnieje możliwość skorzystania z alternatywnego sposobu inicjalizacji kontekstu bezpieczeństwa użytkownika – na podstawie konta użytkownika podanego w formie `"DOMAIN\user"`.

```
/// alternative way of creation of user context
IAzClientContext context
   = application.InitializeClientContextFromName(
      "DOMAIN\user", null);
```

Wadą ostatniego sposobu tworzenia kontekstu użytkownika jest konieczność ustalenia grup, do których należy dany użytkownik, co wymaga wysłania zapytania do kontrolera domeny, co z kolei przekłada się na czas wykonania.

## Authorization Manager (33)

Po utworzeniu kontekstu użytkownika programista może wykonywać kontrolę dostępu w oparciu o metodę **`AccessCheck`** interfejsu **`IAzClientContext`**.

```
class MyApplication {
   const int CHANGE_PASSWORD_OPERATION = 20;
   const int NO_ERROR = 0;
   /// context must be appropriately initialized before use
   IAzClientContext context = null;
   void ChangePasswordAction () {
      object[] operations = { CHANGE_PASSWORD_OPERATION };
      object[] scopes = { "" }; /// use default scope within
                                /// application
      object[] results = (object[])context.AccessCheck(
         scopes, operations, null, null, null, null, null);
      /// get the return code for CHANGE_PASSWORD_OPERATION
      if ((int)results[0] == NO_ERROR) {
         /// access has been granted
      } else { /// access has been denied for some reason
   }
}
```



## Authorization Manager (34)

Za pośrednictem **`AccessCheck`** istnieje również możliwość przekazania parametrów do *BizRule* wywoływanego w trakcie ewaluacji kontroli dostępu.

```
/// based on the code snippet from the previous slide
class MyApplication {
   /// maybe the current user is changing his own password?
   private bool userIsChangingHisOwnPasword() { ... }
   void ChangePasswordActionWithParams () {
      object[] operations = { CHANGE_PASSWORD_OPERATION };
      object[] scopes = { "" };
      /// in alfabethical order
      object[] parNames = { "roles", "self" };
      object[] parValues = { context.GetRoles(),
         userIsChangingHisOwnPassword() };
      object[] results = (object[])context.AccessCheck(
         scopes, operations, parNames, parValues,
         null, null, null);
      if ((int)results[0] == NO_ERROR) { /// access granted
      } else { /// access denied --- check error code }
   }
}
```

# Authorization Manager (35)

```
/// VBScript BizRule corresponding to the code snippet from the
/// previous slide that verifies whether the current user is in
/// the administrator role or wants to change his own password

AzBizRuleContext.BusinessRuleResult = false
isAdministrator = false
roles = AzBizRuleContext.GetParameter("roles")
For Each role in roles
   If role = "Administrator" Then
      isAdministrator = true
      Exit For
   End If
Next
If isAdministrator Then
   AzBizRuleContext.BusinessRuleResult = true
Else
   self = AzBizRuleContext.GetParameter("self")
   AzBizRuleContext.BusinessRuleResult = self
End If
```