Bezpečnost v Javě

Bezpečnost je jedna z vlastností, které bývají u Javy zdůrazňovány jako její silná stránka. Je pravda, že základní bezpečnostní prvky jsou implementovány přímo v jazyce. Tyto bezpečnostní prvky se však musí inicializovat a správně nastavit tak, aby mohly být efektivně využity.

Kromě těchto prvků obsažených přímo v JVM a základních třídách jsou v Javě k dispozici i další knihovny pro zvýšení bezpečnosti aplikací. Jedná se především o zvýšení bezpečnosti při síťové komunikaci a víceuživatelském přístupu. To, co se označuje jako Java security, jsou tři balíčky:

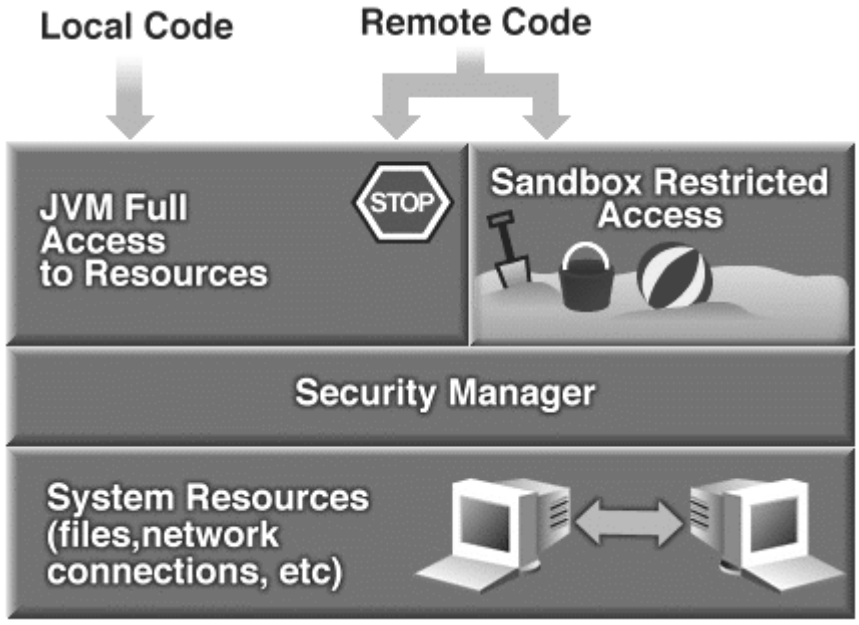
- **JAAS** (Java Authentication and Autorization Service) poskytuje rozhraní, které zajišťuje služby ověřování a autorizace uživatelů pro jednotlivé aplikace.
- **JCE** (Java Cryptography Extention) poskytuje podporu pro šifrovanou komunikaci, výměnu klíčů atd.
- **JSSE** (Java Secure Sockets Extension) poskytuje rozhraní pro programování aplikací, které využívají přenosy po zabezpečené vrstvě SSL.

Popisu těchto balíčků se zde věnovat nebudeme, zájemcům o tuto problematiku mohu doporučit knihu Hacking bez tajemství: Java a J2EE [HackingJava].

Architektura zabezpečení jazyka Java.

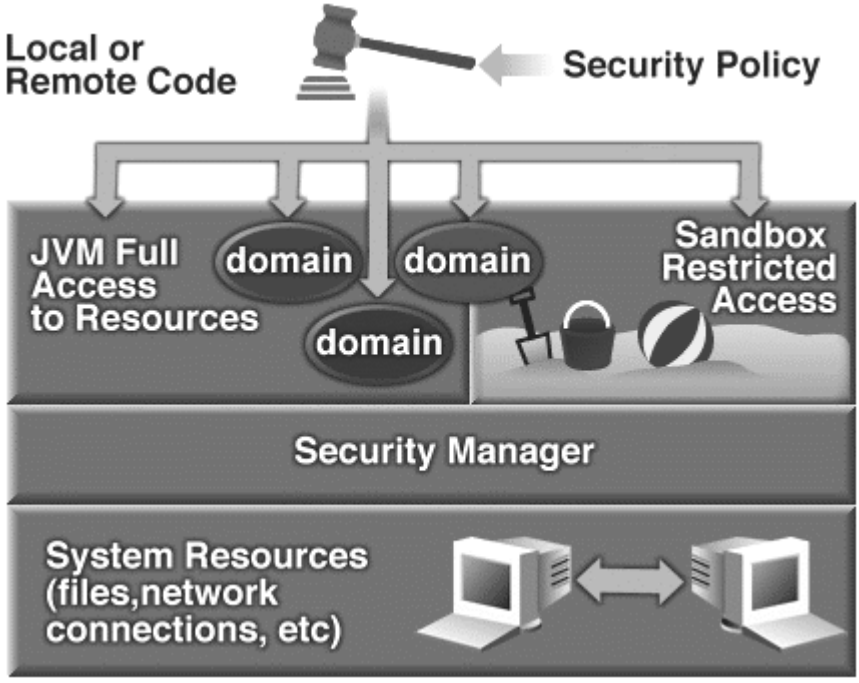
Bezpečnost jazyka Java zajišťuje několik jeho vlastností, např. typová kontrola, hlídání mezí polí, povinnost ošetřovat synchronní výjimky nebo správa paměti. Důležitou součástí zabezpečení je možnost kontrolovat, odkud může aplikace prostřednictvím classloaderu natahovat soubory class a co může kód z různých zdrojů dělat.

Původně byla Java navržena tak, že rozlišovala dva druhy kódu. Kód lokální, který byl uložen a spouštěn lokálně, a kód vzdálený, který byl uložen na síti, stažen a pak lokálně spuštěn (applet). Lokální kód byl důvěryhodný a bylo mu dovoleno vše. Vzdálený kód mohl pracovat pouze v rámci tzv. **bezpečnostního pískoviště (sandbox)**, kde měl velmi omezený přístup k prostředkům. Ve verzi 1.1 navíc přibyla možnost povolit vzdálenému kódu, který byl podepsán důvěryhodnou autoritou, plný přístup k prostředkům. Kontrolu toho, zda je kód důvěryhodný nebo ne, provádí třída **SecurityManager**. Tuto situaci vystihuje Obrázek č. 9.



Obrázek č. 9 Řízení přístupu k prostředkům v Javě 1.1 (převzato z java.sun.com)

Od verze 1.2 poskytuje Java mnohem více možností. Pro kód z různých zdrojů je možno nastavit různé možnosti přístupu ke zdrojům systému. Tato varianta pracuje nad tzv. ochrannými doménami. **Ochranná doména** je definovaná pomocí zásady zabezpečení (security policy). Bezpečnostní pískoviště je vlastně také doménou s pevně danými hranicemi. Takto řízený přístup k prostředkům je ukazuje Obrázek č. 10.



Obrázek č. 10 Řízení přístupu k prostředkům v Javě verze 1.2 a vyšší (převzato z java.sun.com)

Veškerý kód, který je součástí SDK, je považován za důvěryhodný a má nastavena veškerá oprávnění. Aplikační kód není standardně spouštěn pomocí správce zabezpečení, pokud správce zabezpečení ve své aplikaci spustíte, nebude mít vaše aplikace žádná práva (kromě

práva na čtení souborů z adresáře, ve kterém je uložena, a všech jemu podřízených). Pro nastavení práv pak musíte k aplikaci přiřadit soubor zásad zabezpečení, ve kterém popíšete, co je aplikaci dovoleno. Pokud spouštíte v prohlížeči stránku, která obsahuje applet, je automaticky spuštěn i správce zabezpečení a applet může fungovat pouze v rámci bezpečnostního pískoviště.

Jestliže je spuštěn bezpečnostní manager, každý požadavek na získání systémových zdrojů (např. otevření souboru, vytvoření spojení v síti, stažení souboru ze sítě) vyvolá několik akcí. JVM spustí metodu **SecurityManager.checkPermission()**, ta zavolá metodu **checkPermission()** instance třídy **AccessController**. Tato metoda v souboru zásad zabezpečení zkontroluje, zda má tento kód k tomuto zdroji přístup. Pokud přístup není dovolen je vyhozena výjimka typu **SecurityException**.

Součástí standardní instalace Javy je soubor **java.security**, který definuje základní umístění zásad zabezpečení a možnost jejich rozšiřování. Pro nás jsou důležité tyto jeho součásti:

```
policy.url.1=file:${java.home}/lib/security/java.policy
policy.url.2=file:${user.home}/.java.policy

policy.expandProperties=true
```

První dva řádky říkají, kde se mají hledat soubory zásad zabezpečení a poslední řádek povoluje přidávat soubory zásad zabezpečení i odjinud. Soubor zabezpečení, který je standardní součástí Javy je tedy uložen ve stejném adresáři a jmenuje se **java.policy**. Obsah tohoto souboru vidíte v následujícím výpise. Povoluje třídám standardu Javy všechno a pro ostatní třídy nastavuje několik možností pro čtení.

```
// Standard extensions get all permissions by default

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

// default permissions granted to all domains

grant {
permission java.lang.RuntimePermission "stopThread";
// allows anyone to listen on un-privileged ports
permission java.net.SocketPermission "localhost:1024-","listen";
// "standard" properies that can be read by anyone
permission java.util.PropertyPermission "java.version", "read";
permission java.util.PropertyPermission "java.vendor", "read";
permission java.util.PropertyPermission "java.vendor.url","read";
permission java.util.PropertyPermission"java.class.version", "read";
permission java.util.PropertyPermission "os.name", "read";
permission java.util.PropertyPermission "os.version", "read";
permission java.util.PropertyPermission "os.arch", "read";
permission java.util.PropertyPermission "file.separator","read";
permission java.util.PropertyPermission "path.separator","read";
permission java.util.PropertyPermission "line.separator","read";
```

```
permission java.util.PropertyPermission
"java.specification.version","read";
permission java.util.PropertyPermission
"java.specification.vendor","read";
permission java.util.PropertyPermission
"java.specification.name","read";
permission java.util.PropertyPermission
"java.vm.specification.version", "read";
permission java.util.PropertyPermission
"java.vm.specification.vendor", "read";
permission java.util.PropertyPermission
"java.vm.specification.name", "read";
permission java.util.PropertyPermission "java.vm.version", "read";
permission java.util.PropertyPermission "java.vm.vendor", "read";
permission java.util.PropertyPermission "java.vm.name", "read";
};
```

Pokud nezměníme nastavení vlastnosti policy.expandProperties na false, může jakákoli aplikace načíst ještě další soubory zásad zabezpečení. Jak tyto soubory vytvořit a co do nich lze zapsat si ukážeme v následující kapitole. Jaké soubory zásad zabezpečení má SecurityManager použít lze nastavit dvěma způsoby.

- Je možné přímo do souboru java.security přidat URL dalšího souboru zásad zabezpečení.
- Při spouštění aplikace je možné nastavit systémovou vlastnost java.security.policy. Když při přiřazování použijete jedno rovnítko, bude uvedený soubor použit společně se soubory uvedenými v souboru java.security. Jestliže použijete dvě rovnítka, bude použit jen uvedený soubor zásad zabezpečení.

Spuštění správce zabezpečení je velmi jednoduché.Nejprve bychom si měli ověřit, že již není nějaký správce zabezpečení používán, a pak pomocí statické metody setSecurityManager() třídy System přiřadit aplikaci správce. Jak to zapsat do kódu vidíte v následujícím výpisu.

```
.............
SecurityManager manager = System.getSecurityManager();
if (manager == null) {
    System.setSecurityManager(new SecurityManager());
}
.............
```

Jak se vytvářejí soubory zásad zabezpečení.

Soubor zásad zabezpečení je soubor ve formátu prostého textu, je tedy možné ho vytvářet a editovat pomocí textového editoru. Java jako součást SDK poskytuje nástroj **policytool**. Má grafické rozhraní a je možné ho spustit z příkazové řádky pomocí příkazu policytool.

Soubor zásad zabezpečení se skládá z posloupnosti záznamů udělujících oprávnění. Nepovinně může být v souboru uvedeno úložiště certifikátů, kde je možno hledat certifikáty a jejich klíče. Pak následuje alespoň jeden záznam **grant**. Za záznamem grant může následovat klauzule **SignedBy**, klauzule **CodeBase**, klauzule **Principal** nebo jejich kombinace. Uvedeme li klauzuli SignedBy a za ní seznam autorit, získá kód podepsaný těmito autoritami dále nastavená oprávnění. Pokud uvedeme klauzuli CodeBase a za ní URL, říkáme, že kód natažený z tohoto URL získá dále uvedená oprávnění. Klauzule Principal slouží k nastavení oprávnění pro uživatele a je využíváno balíčkem JAAS. Pokud není ani jedna klauzule uvedena, následující oprávnění nastavujeme pro všechen kód aplikace.

Každý záznam grant obsahuje jedno nebo více oprávnění. Oprávnění jsou potomky třídy java.security.Permission. Popíšeme si nejčastěji používané třídy oprávnění.

AllPermission

Toto oprávnění uděluje kódu právo vykonávat všechny operace. Znamená to tedy, že pokud udělíte kódu toto oprávnění, je to stejné jako když nepoužijete správce zabezpečení. Následující ukázka souboru zásad zabezpečení tedy znamená „všichni mohou všechno“.

```
grant {
    permission java.security.AllPermission;
};
```

Toto oprávnění nedoporučuji vůbec používat. Bylo vytvořeno proto, aby mohla být jednoduše nastavena oprávnění pro třídy JRE viz soubor java.policy v instalace SDK.

FilePermission

Tato třída slouží k nastavení oprávnění pro přístup k souborům a adresářům. Toto oprávnění má dva parametry, první určuje soubor či adresář, druhý pak povolené operace. Můžeme povolovat tyto operace:

- read
- write
- execute
- delete

Při určování adresářů a souborů na které se oprávnění vztahuje je možné použít i znak * a -, jejich význam si ukážeme na následujícím příkladě.

```
grant{
    permission java.io.FilePermission “C:\\java\\*”,”read”;
    permission java.io.FilePermission “C:\\temp\\-“,”read, write”;
    permission java.io.FilePermission “<<ALL FILES>>”,”read”;
    permission java.io.FilePermission “*”,”read, delete, write”;
    permission java.io.FilePermission “-”,”read, execute”;
};
```

První oprávnění říká, že kód má oprávnění číst všechny soubory z adresáře java na disku C. Druhé oprávnění povoluje čtení, vytváření a upravování souborů a adresářů, které jsou v adresáři temp na disku C a ve všech jeho podadresářích. Třetí oprávnění říká, že je možné číst všechny soubory a adresáře v souborovém systému. Čtvrté oprávnění umožňuje číst, mazat, vytvářet a upravovat všechny soubory v aktuálním adresáři. Poslední oprávnění říká, že je možné číst a spouštět všechny soubory z aktuálního adresáře a všech podřízených.

Pozor na používání oddělovačů v zápisu cesty k souborům a adresářům. Používejte ten oddělovač, který používá aktuální operační systém. Pro Windows tedy znak \ a pro Linux /. Podle mých zkušeností použití / ve Windows vede k tomu, že nastavení oprávnění neproběhne a aplikace vyhazuje SecurityException. Zdvojení \ je nutné proto, že parametr oprávnění se zapisuje a zpracovává jako instance třídy String.

SocketPermission

Třída java.net.SocketPermission slouží k nastavení oprávnění pro vytváření socketových připojení. Oprávnění má dva parametry a to název hostitele a port a seznam povolených akcí. Akce může být:

- accept – povoluje přijetí spojení
- connect – je povoleno vytvořit spojení
- listen – umožňuje pouze sledovat příchozí spojení, ne je přijmout
- resolve – umožňuje překlad názvu DNS na IP adresu a naopak, toto oprávnění v sobě zahrnují všechna tři předchozí

První parametr se uvádí ve formátu název hostitele:číslo portu. Jako název hostitele je možné uvést název DNS, IP adresu nebo hodnotu “localhost“. V názvu je možné použít znak * a při určování portů je možné uvést i interval. V následujícím příkladu si ukážeme některé možnosti nastavení SocketPermission.

```
grant{
  permission java.net.SocketPermision “java.vse.cz:1234-“,“accept”;
  permission java.net.SocketPermision “*.vse.cz”,”connect”;
};
```

První oprávnění říká, že je možné přijmout připojení z java.vse.cz na portech s číslem 1234 a vyšším.

Druhé oprávnění znamená, že je možné vytvářet spojení na všechny počítače z domény vse.cz a to přes jakýkoli port.

Java poskytuje ještě několik tříd pro nastavení oprávnění např. java.awt.Permission, java.util.PropertyPermission nebo java.lang.RuntimePermission.