|  |
| --- |
| **Statefullness** |
| Stateless | Stateful: Object pools and/or lots of new objects |
| **Configuration** |
| More flexible and secure configuration is needed for better operational support at the enterprise level and to provide developers more flexibility in using ESAPI without having to tweak ESAPI.properties for minor changes.(Mike: A thousand pardons for inserting this here, but I didn't know where else to place this. Also, apologies for messing up the formatting of the previous row. --kevin) | My vision for this is to do something like this. Pardon me for using crypto as the example. dfltEncryptor = ESAPI.encryptor(); Properties tDesProps = new Properties(); props.setProperty("Encryptor.CipherTransformation", "DESede/CBC/PKCS5Padding"); props.setProperty("Encryptor.EncryptionKeyLength", "128"); tdesEncryptor = ESAPI.encryptor(tDesProps);If for each "component" in ESAPI class, we have two interfaces, e.g., encoder() and encoder(Properties) authenticator() and authenticator(Properties) etc.Then if you want all defaults from ESAPI.properties (or wherever), use the first form. You use the 2nd for two apply your properties to override any defaults or add your own propers specific to your implementation.The only thing that I'd want on top of this is that they needs to be a separation of properties. There need to be some that an operational team can set and that the dev team absolutely cannot override. (Although, that probably may requires a SecurityManager in Java). But this is important for enterprises where operations teams help corporate security to enforce security policies of deployed software. Also, would be nice to specify some properties used by ESAPI in secure fashion. Mimimally, I'd like to see support for a signed properties file to prevent tampering. Operations team could sign it and ESAPI could verify it.  |
| **Typing** |
| Weakly typed in 1.4 orGenerics allow strong typing in 1.5+ | Strongly typed |
| **Integration with code** |
| Configurable in properties files or hard coded into code | Hard coded into code |
| **Exception Handling** |
| Centralized uncaught exception handling forces exceptions to fail closed. | Developer is responsible for exception handling |
| **What a call looks like in the code (Manual Integration)** |
| AccessController.isAuthorized("SpecialPolicy");AccessController.isAuthorized("SpecialPolicy1", data); | AccessController.isAuthorized("DELETE\_FILE", new DeletePolicy("filename")); |
| **What a developer implements** |
| public class SpecialPolicy1 extends ESAPIPolicy{ public boolean evaluate(Object o) { //special rules go here return t/f; }} | public class DeletePolicy extends ESAPIPolicyCollection{ protected String filename; public DeletePolicy(String filename) { this.filename = filename; }  public boolean evaluate (String data) { //filename validation rules  //return true or false. }} |
| **Integration** |
| **Manual Integration Option**AccessController.isAuthorized("SpecialPolicy");AccessController.isAuthorized("SpecialPolicy1", data);**Integration Option**@policy KeyPolicyFile@policy MethodToKeyPolicyFile@policy AspectJ <cut stuff>@Policy JSR115@key StandardSimplePolcypublic class StandardSimplePolcy{ public boolean evaluate(Object<T> data) throws Exception { //cast data //do file look up stuff  }}**Properties file Integration**policy.propertiesSpecialPolicy=com.foo.model.policy.SpecialPolicySpecialPolicy1=com.foo.model.policy. StandardSimplePolcycom.mypackage.myclass.mymethod=SpecialPolicy | **Manual integration**AccessController.isAuthorized("DELETE\_FILE", new DeletePolicy("filename")); |