Security Pattern Catalog Schema: Common Structure.

(schema/SecurityPatternCatalogNaiveSchema.owl)

Andrei Brazhuk (brazhuk@grsu.by)

Property	*Range and predefined values*	*Description*	*Example*
Class: **Pattern** Class: SecurityPattern (Defined class) Class: ThreatPattern (Defined class) Class: MisusePattern (Defined class)			*pattern_SecureDistributedPublishSubscribeIoT*
Metadata: * To describe a pattern you should use the “textIntent”, “textProblem”, “textSolution*”, “textConsequences”, “textImplementation” fields. And in some cases it is impossible to put a full description of the pattern because of copyright and trademark, so the "textreview*" fields retell a content of the pattern in own words. In the second case links to pattern’s original document should be provided.			
textName	**xsd:String**	Pattern’s primary name	*Secure Distributed Publish/Subscribe (P/S) pattern for IoT*
textAKAName	**xsd:String**	Pattern’s alternative name(s)	
textAuthor	**xsd:String**	Pattern’s author(s)	*Eduardo B. Fernandez* *Nobukazu Yoshioka* *Hironori Washizaki*
textURL	**xsd:String**	URL(s) of webpage that describes a pattern	*https://www.researchgate.net/publication/339103887_Secure_Distributed_PublishSubscribe_PS_pattern_for_IoT*
textPDF	**xsd:String**	Downloadable URL(s) of pattern’s PDF	
textReference	**xsd:String**	A bibliographic description of a primary paper, describing a pattern	*E.B.Fernandez, N. Yoshioka, H. Washizaki, “Secure Distributed Publish/Subscribe (P/S) for IoT, 2020. Procs. Asian PLoP'20, March 4 6, Taipei, Taiwan. 9 pages.*
textReviewContext	**xsd:String**	A brief text description of pattern’s context	Something like that: *Information exchange between IoT/IIoT devices (e.g. smart thermostats or sprinkler systems,*

			different sensors) with minimal security control and cloud/fog applications.
textReviewProblem	**xsd:String**	A brief text description of pattern's problem	Something like that: *Subscribers (S) register and receive messages of their interest sent by a publisher (P). The main concerns are how to organize the interactions between them securely, avoiding rogue participants, insecure communications, unwanted P/S operations.*
textReviewSolution	**xsd:String**	A brief text description of pattern's solution	*In addition of the standard P/S functions it is possible to use secure channels for protected communications, access control for restricting the actions of publishers and subscribers, security logging, and digital signatures.*
textIntent	**xsd:String**	Pattern's Intent (full text)	*In an IoT system, decouple the publishers of events from those interested in the events (subscribers). Subscription and publication are performed securely.*
textContext	**xsd:String**	Pattern's Context (full text)	
textProblem	**xsd:String**	Pattern's Problem (full text). It includes forces that constrain the solution.	
textSolution	**xsd:String**	Pattern's Solution description.	
textSolutionStructure	**xsd:String**	It describes the structure (static view) of the solution and some dynamic aspects in the form of sequence diagrams for a use case.	
imgSolutionStructure	**xsd:String**	Downloadable URL(s) of a solution structure diagram(s).	
textSolutionDynamics	**xsd:String**	It describes the dynamic aspects of the solution with diagrams.	
imgSolutionDynamics	**xsd:String**	Downloadable URL(s) of a solution	

		dynamics diagram(s).	
textImplementation	**xsd:String**	The objective of this section is to describe what one should consider when implementing the pattern. This can be a set of general recommendations, or a sequence of what to do to use the pattern. It may include some sample code, if appropriate.	
textConsequences	**xsd:String**	This section indicates the BENEFITS and LIABILITIES of the solution embodied in this pattern. The benefits should match the forces in the Problem section. Benefits that do not correspond to any force may appear.	
textKnownUses	**xsd:String**	To accept a solution as a pattern, it should be found at least three examples of its use in real systems.	
Organization and scope:			
hasType	**Type** Predefined items: **type_SecurityPattern** **type_ThreatPattern** **type_MisusePattern**	Type of a pattern.	*hasType value type_SecurityPattern*
hasTemplate	**Template** Predefined items: **template_POSA** **template_GOF**	Template, used to describe a pattern. It can be POSA or GOF. POSA stands from “Pattern Oriented Software Architecture”. GOF stands from “Gang of Four”.	*hasTemplate value template_POSA*
hasGroup (inverse: isGroupOf)	**Group**	Tells to which group a pattern belongs to.	*hasGroup value patterngroup_SecureMiddleware*

usesPattern (inverse: isUsedBy)	**Pattern**	Enumerates patterns that are used by this one.	*pattern_RoleBasedAccessControl* *pattern_Authenticator* *pattern_SecurityLoggerAuditor* *pattern_SecureChannel*
relatesTo (symmetric)	**Pattern**	Enumerates patterns that are related to this one.	*pattern_SecurePS* *pattern_Broker* *pattern_SecureChannel* *pattern_EnterpriseServiceBus* *pattern_Authorizer* *pattern_IoTSegmentation*
isChildOf (inverse: isParentOf)	**Pattern**	For a concrete pattern shows from which abstract pattern it has been made. It can be possible to use the class assertion here, e.g. create a hierarchy with abstract patterns on the top and concrete ones at the bottom.	*IsChildOf value* *pattern_SecurePublishSubscirbe*

Common characteristics:

hasDomain	**Domain** Predefined items: **domain_FogComputing** **domain_EdgeComputing** **domain_InternetOfThings** **domain_SCADA** **domain_Military** **domain_ECommerce** **domain_GridComputing** Class: CloudComputingDomain ***domain_CloudComputing*** **domain_IaaS** **domain_PaaS** **domain_SaaS** **domain_NFV**	Tells to which domain(s) a pattern belongs to. Domain is a large functional field of Information Technologies (IT), like Cloud Computing, Internet of Things. It might be less gigantic, like IaaS or NVF.	*hasDomain value* *domain_InternetOfThings*
hasArchitecturalLayer	**ArchitecturalLayer**	Shows a common architectural domain, to which a pattern relates, like	*hasArchitecturalLayer* *value al_ClientLayer*

	Predefined items: Class: ApplicationArchitecturalLayer **al_ClientLayer** **al_LogicLayer** **al_DataLayer** Class: PlatformAndOperatingSystemLayer **al_PlatformAndOperatingSystem** Class: CommunicationArchitecturalLayer **al_DistributionLayer** **al_TransportLayer** **al_NetworkLayer**	Applications, Platform and Operating systems, also Communications [Vale, 2019]. Instances are taken from [VanHilst, 2009]	*hasArchitecturalLayer value al_DistributionLayer*
hasConstraintLevel	**ConstraintLevel** Predefined instances: **cl_RegulatoryLevel** **cl_OrganizationalLevel** **cl_HumanLevel** **cl_MechanismLevel**	Refers to four levels of constraint: mechanism, human (operator or developer), organizational, and regulatory(Leveson, 2004). Instances are taken from [VanHilst, 2009]	*hasConstrainLevel value cl_MechanismLevel*
hasResponseType	**ResponseType** Predefined instances: **rt_Avoidance** **rt_Deterrence** **rt_Prevention** **rt_Detection** **rt_Mitigation** **rt_Recovery** **rt_Forensics**	"The response axis based on whether or not and attack happens and the extent, from not happening at all (avoidance), to completely happened and in the past (forensics)." [VanHilst, 2009] Instances are taken from [VanHilst, 2009]	*hasResponseType value rt_Avoidance*
hasLifecycleStage	**LifecycleStage** Predefined instances: **lc_ArchitectureAndDesign** **lc_BuildAndCompilation** **lc_Implementation** **lc_Installation** **lc_Operations** **lc_Requirements**	Tells which which system's lifecycle stage a pattern is applicable. Frankly, most of the patterns are applicable on the Design (Architecture) stage, but it might be possible to have a few exceptions. Instances are taken from [CAPEC].	*hasLifecycleStage value lc_ArchitectureAndDesign*

	lc_SystemConfiguration **lc_Deployment**		
hasSecurityLevel	**SecurityLevel** Predefined instances: **sl_PhysicalSecurity** **sl_PersonnelSecurity** **sl_CommunicationAndDataSecurity** **sl_OperationalSecurity**	Tells to which field of security a pattern belongs to. To review. https://en.wikipedia.org/wiki/Physical_security https://en.wikipedia.org/wiki/Secure_communication https://en.wikipedia.org/wiki/Communications_security https://en.wikipedia.org/wiki/Operations_security	*hasSecurityLevel value* *sl_CommunicationAndDataSecurity*

Context characteristics:

Domain metamodel			
suggestsPart **hasPart** (transitive) **isPartOf** (inferred, inverse to hasPart)	**Component**	It suggests a component consists from other components (concept level) An instance “hasPart” another instance. An instance “isPartOf” another instance.	
suggestsInteraction **interacts** (symmetric)	**Component**	It suggests a component interacts with another component (concept level) An instance “interacts” with another instance (sym)	
produces **isProducedBy** (inferred, inverse to produces)		An instance “produces” another instance. An instance “isProducedBy” another instance	

suggestsProduction	**Component**	It suggests a component produces another component (concept level)	
suggestsFunction **hasFunction** **isFunctionOf** (inferred, inverse to hasFunction)		It suggests a component "hasFunction" some function An instance "hasFunction" some function	
Main context parameters			
hasAffectedFunction	**Function** Predefined classes & instances: Class: **ActorFunction** **function_Actor** Class: **HardwareFunction** **function_Hardware** Class: **SoftwareFunction** **function_Software**	Tells which system function(s) a pattern affects.	*hasAffectedFunction value function_DistributeEventInformation* *hasAffectedFunction value function_DistributeSensorData*
hasAffectedComponent	**Component** Predefined classes & instances: Class: **ActorComponent** **component_Actor** Class: **HardwareComponent** **component_Hardware** Class: **SoftwareComponent** **component_Software**	Tells which common component(s) a pattern affects.	*hasAffectedComponent value component_IoTApplication* *hasAffectedComponent value component_IIoTApplication* *hasAffectedComponent value component_CloudApplication* *hasAffectedComponent value component_FogApplication*
Inferred context parameters			
isAffectedFunctionOf (inferred, inverse to hasAffectedFunction)			-
isAffectedComponentOf		Component is affected by a pattern	-

(inferred, inverse to hasAffectedComponent)		Режим декомпозиции (предлагаются только шаблоны непосредственно привязанные к данному компоненту)	
isAffectedComponentOf ViaFunction (inferred)	**hasFunction o isAffectedFunctionOf** (property chain)	Component is affected by patterns, which affect its function	-
isAffectedComponentOf ViaPart (inferred)	**hasPart o isAffectedComponentOf** (property chain)	Component is affected by patterns which affect parts of the component (hasPart is transitive) Режим монолитный (предлагаются шаблоны, которые привязаны к различным возможным частям этого элемента)	

Security characteristics:

hasSecurityConcern	**SecurityConcern** Predefined instances: concern_AccessControl concern_Awareness concern_Training concern_Audit concern_Accountability concern_SecurityAssessment concern_Authorization concern_ConfigurationManagement concern_ContingencyPlanning concern_IdentificationAndAuthentication concern_IncidentResponse concern_Maintenance concern_MediaProtection concern_PhysicalProtection concern_EnvironmentalProtection concern_Planning concern_PersonnelSecurity	Tells which security concern(s) a pattern touches. “A security concern represents some security feature(s)” [Guan, 2016]. Instances are taken from [NIST SP 800-53].	*hasSecurityConcern value concern_AccessControl* *hasSecurityConcern value concern_Audit* *hasSecurityConcern value concern_InformationIntegrity* *hasSecurityConcern value concern_CommunicationsProtection*

	concern_RiskAssessment concern_ServicesAcquisition concern_CommunicationsProtection concern_SystemProtection concern_SystemIntegrity concern_InformationIntegrity concern_ProgramManagement		
hasThreat (inverse: isThreatOf)	**Threat** Predefined instances: see ***schema_threats.pdf*** Class: CommunicationsThreat **threat_ManInTheMiddle** **threat_Interception** **threat_Flooding** **threat_ContentSpoofing** **threat_IdentitySpoofing** **threat_Footprinting** **(=threat_InformationGathering)** **threat_ProtocolAnalysis** Class: SoftwareThreat **threat_SessionManipulation** **threat_AuthenticationBypass** **threat_PriviledgeEscalation** **threat_Excavation** **threat_CodeInjection** **threat_BufferManipulation** **threat_ExcessiveAllocation** **threat_ManipulationAPI** **threat_InputDataManipulation** **threat_EnvironmentManipulation** **threat_SharedDataManipulation** **threat_Malware**	Tells what threats a pattern describes with connection to component(s) and function(s), figured out on the previous stage. For security patterns defines the possible threats, met by a pattern. For attack pattern defines the possible threats, produced by an implementation of pattern. Instances are adopted from [CAPEC] (see ***schema_threats.pdf***)	*hasThreat value threat_PriviledgeEscalation* *hasThreat value threat_IdentitySpoofing* *hasThreat value threat_Interception* *hasThreat value threat_ContentSpoofing* *hasThreat value threat_Flooding*
Inferred security characteristics:			

hasInferredSTRIDE	**hasThreat o hasSTRIDE**	Pattern's STRIDE	
hasInferredSecurityObjective	**hasThreat o hasSecurityObjective**	Pattern's SO	
hasInferredThreatImpact	**hasThreat o hasThreatImpact**	Pattern's threat impact	
hasPossibleAttack	**Attack**	Will be taken from [CAPEC] and other attacks' classifications.	
hasPossibleWeakness	**Weakness**	Will be taken from [CWE] and other weaknesses'/vulnerabilities' classifications.	

Class: **Threat**

Contains automatically assigned properties. Automatic reasoning procedures will get them from the internal data scheme.

hasThreatImpact (inverse: isThreatImpactOf)	**ThreatImpact** Predefined instances: see ***schema_threats.pdf*** ti_AlterExecutionLogic ti_BypassProtectionMechanism ti_ExecuteArbitraryCode ti_GainPrivileges ti_HideActivities ti_ModifyData ti_ReadData ti_ResourceConsumption ti_UnreliableExecution	Tells which negative impact(s) the threats, described by a pattern, have to component(s) and function(s). It obtains from the CAPEC attack descriptions (the hasThreat property here).	
hasSTRIDE (inverse: isSTRIDEof)	**STRIDE** Predefined instances: STRIDE_Spoofing STRIDE_Tampering STRIDE_Repudiation STRIDE_Information_Disclosure STRIDE_Denial_of_Service STRIDE_Elevation_of_Privilege	Tells which STRIDE item(s) a pattern touches. To do: map STRIDE & SO	
hasSecurityObjective (inverse:	**SecurityObjective**	Tells which security objective(s) a pattern touches.	

isSecurityObjectiveOf)	Predefined instances: SO_AccessControl SO_Accountability SO_Authentication SO_Authorization SO_Availability SO_Confidentiality SO_Integrity SO_NonRepudiation		
Support			
	SchemaInstance	Holds instances which belong to this ontology	
	SchemaStub	Holds instances which have a defined class.	

References:

[Guan, 2016] H. Guan, H. Yang, and J. Wang, “An ontology-based approach to security pattern selection,” Int. J. Autom. Comput., vol. 13, pp. 168–182, Apr. 2016.

[Vale, 2019] A.P. Vale, E. B. Fernández, “An Ontology for Security Patterns”. Conference paper. 2019.

[VanHilst, 2009] VanHilst M. et al. A multi-dimensional classification for users of security patterns //Journal of Research and Practice in Information Technology. – 2009. – Т. 41. – №. 2. – С. 87.

[CAPEC] https://capec.mitre.org/

[CWE] https://cwe.mitre.org/

[Fernandez, 2013] E. B. Fernandez, Security patterns in practice: designing secure archi- tectures using software patterns. John Wiley and Sons, 2013.

[NIST SP 800-53] Security and privacy controls for federal information systems and organizations NIST Special Publication 800-43 revision 4 //NIST. – 2013.