Business Process Dependencies and Inference Rules Formal Definitions

Gal Engelberg[1,2][0000−0001−9021−9740], Moshe Hadad[1,2][0000−0002−9315−6260], and Pnina Soffer[1][0000−0003−4659−883X]

[1] University of Haifa, Abba Khoushy Ave 199, Haifa, 3498838
spnina@is.haifa.ac.il
[2] Accenture Labs, Tel Aviv, Israel
{moshe.hadad,gal.engelberg}@accenture.com

1 Running Example

The business process model depicted in Figure 1 is a credit request evaluation process, introduced by [4]. The process model outlines the steps involved in assessing, verifying, and approving or rejecting credit applications. The process initiates when a credit request is received. The Verify task examines the requester's information and the requested amount. Based on the verification result and amount threshold (above or below 5000), the request proceeds to either an Advanced Assessment for high amounts or a Simple Assessment for lower amounts. Each assessment yields a decision, which, if unfavorable, prompts a Renegotiate Request task; otherwise, it moves to the final decision stage.

Data elements such as verification status, amount, interest, requester's first name initial, and final decision are used to guide the process flow. Decision points (gateways) are utilized to ensure the workflow conforms to specified business rules, including notification handling if the decision is negative.

The workflow leverages specific resources, with roles defined as Assistants and Experts. Assistants (Pete, Mike, Ellen) handle initial verification and customer notifications, while Experts (Sue, Sean) conduct assessments. A Role Manager (Sara) oversees the approval stage if the request is successful. This structured use of data and resource allocation ensures a controlled and efficient credit evaluation process.

2 Business Process Model Dependencies

In order to explain the cascading effects of cyber loss events within business processes, this study aims to analyze their impact on the following process model entities: resources, activities, events, and gateways. Our analysis targets following types of dependencies in the process model: *1) Resource-to-Activity:* we investigate how resource participation in a risk event affects the related activities. *2) Control Flow:* we assess the impact of risk events involving activities, events, and gateways on the subsequent activities, events and gateways within the process. *3)Data Flow:* we analyze how the involvement of data resources in a

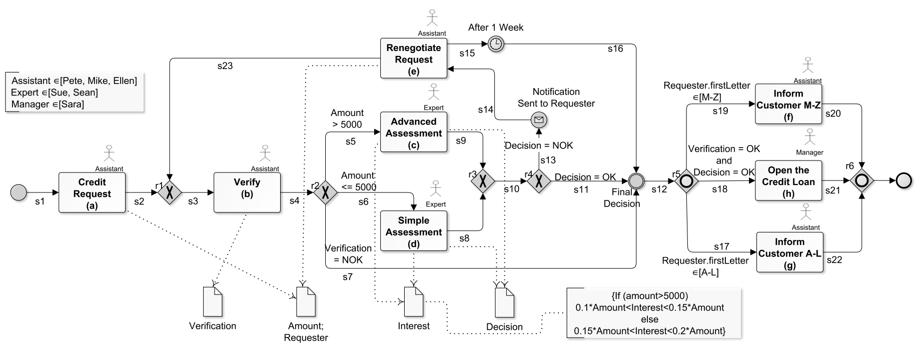

Fig. 1. A data-and-resource-aware process model of a credit evaluation process, as introduced in [4]

risk event influences other entities within the process. Since resource-to-resource dependencies are frequently not explicitly represented in process models and can encompass a broad spectrum of domain-specific dependencies, such as those found in software component relationships [8], we assume that all dependent objects function as input resources for an activity. For instance, when an activity is conducted by a human utilizing a computer, we assume that both the human and the computer are considered resources for that activity. In this section, we provide a precise definition of these types of dependencies based on the BBO model illustrated in Figure 2. To facilitate this, we utilize formal logic, where predicates represent relationships, and the arguments correspond to entities within the UML model.

Resource-to-Activity Dependencies According to [2], resources within a process are either produced or consumed by activities. A resource is produced when an activity generates outputs, such as data or items, that are necessary for subsequent tasks, for example, creating a report or assembling a product component. Conversely, a resource is consumed when an activity uses it as input, modifying or utilizing it to achieve its objectives, as in the case of utilizing raw materials in manufacturing or processing data for analysis. As noted by [5], the primary dependencies between activities and resources are OR and AND. In an OR dependency, if an activity a_1 depends on resources r_1 or r_2, a_1 can proceed if either r_1, r_2, or both are available. Conversely, in an AND dependency, all required resources must be simultaneously available for the activity to proceed.

In the BBO representation, dependencies between activities and resources are organized into input and output sets. The input set of an activity contains resources necessary to initiate and execute that activity, while the output set includes resources that are generated or modified upon the activity's completion. These resources may include various elements, such as data, documents, personnel, or equipment.

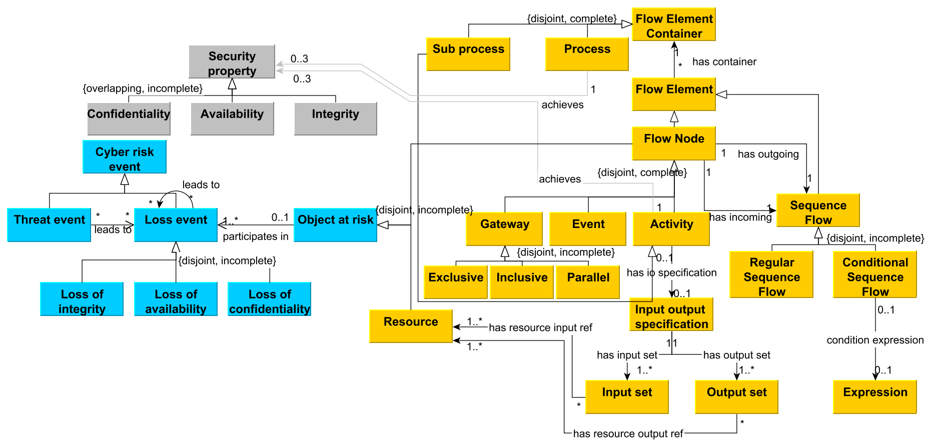

Fig. 2. A UML diagram represents the business process model, its components, and key security properties. Yellow elements are derived from BBO[3], blue elements are inspired by COVER [9], grey elements are inspired by the secure business process definition in [6].

The dependencies of input sets are expressed in equation 1. In our example, the activity required to conduct an *Advanced Assessment* has two primary inputs: a human resource with the role of *Expert* and a data resource representing the credit request *Amount*. All human resources designated as *Experts* are included within a single input set, while the *Amount* data resource constitutes a separate input set. For the activity to proceed, all input sets must be available, reflecting an AND dependency between resources in different input sets, whereas an OR dependency applies within a single input set, indicating that only one resource in that set needs to be available.

$$\begin{aligned}\forall a \forall ios \forall is \forall r\ \big(\, & HasIoSpecification(Activity(a), InputOutputSpecification(ios)) \wedge \\ & HasInputSet(InputOutputSpecification(ios), InputSet(is)) \wedge \\ & HasResourceInputRef(InputSet(is), Resource(r)) \rightarrow \\ & InInputSet(r, a)\,\big)\end{aligned} \tag{1}$$

Similarly, resources in the output set of an activity are represented as shown in 2. For example, after the *Advanced Assessment* activity, output resources might include data elements like *Interest* and *Decision*. Notably, the *Decision* data resource serves as an input for subsequent activities, such as *Renegotiate Request*. These dependencies are formalized in the referenced equation.

$$\begin{aligned}\forall a, \forall ios, \forall os, \forall r\ \big(& HasIoSpecification(Activity(a), InputOutputSpecification(ios)) \wedge \\ & HasOutputSet(InputOutputSpecification(ios), OutputSet(os)) \wedge \\ & HasResourceOutputRef(OutputSet(os), Resource(r)) \rightarrow \\ & InOutputSet(r, a)\,\big)\end{aligned} \tag{2}$$

Control Flow Dependencies Control flow dependencies are defining the logical sequence for executing activities in a business process. As [11] describes, control flow patterns fall into two primary categories: synchronized and unsynchronized. Synchronized patterns require multiple process branches to reach specific points simultaneously, ensuring that activities converge before the next stage can proceed. In contrast, unsynchronized patterns allow branches to operate independently, where each branch can advance without needing others to complete. Our study focuses on synchronized control flow patterns, specifically examining the core patterns identified by [11]: Normal Sequence Flow, Exclusive, Inclusive, and Parallel control-flow patterns.

The Normal Sequence Flow pattern is the most straightforward, as it dictates a linear progression where one flow node directly follows another. It is limited to flow nodes, which can be activities or events, distinguishing it from more complex patterns that involve gateways. In our model, these dependencies expressed in equation 3:

$$\begin{aligned}\forall a_1, \forall a_2, \forall s \big(&HasOutgoing(FlowNode(a_1), SequenceFlow(s))\wedge\\ &HasIncoming(FlowNode(a_2), SequenceFlow(s))\wedge\\ &(s \in NormalSequenceFlow)\wedge\\ &(a_1 \in \{Event, Activity\}) \wedge (a_2 \in \{Event, Activity\})\\ &\rightarrow NormalSequenceFlow(a_1, a_2)\big)\end{aligned} \tag{3}$$

The Exclusive pattern selects one path from multiple options based on specific conditions, with only the chosen branch executed. Once it completes, the process resumes at the join without waiting for other branches. The Inclusive pattern allows multiple branches to run simultaneously if conditions are met, pausing at the join until all selected branches finish. The Parallel pattern activates all branches at once, with the process waiting at the join until every branch completes. This pattern is ideal for coordinating parallel tasks that require synchronization.

In our model, these patterns are described as instances of input flow nodes (a_1), and output flow nodes (a_2), connected to the gateway (g) by instances of input sequence flow (s_1) and output sequence flow (s_2), where the output sequence flow might have a conditional expression which routes the control flow according to data resource values. These dependencies expressed in equation 4:

$$\begin{aligned}\forall a_1, \forall a_2, \forall g, \forall s_1, \forall s_2, \forall e \big(&HasOutgoing(FlowNode(a_1), SequenceFlow(s_1))\wedge\\ &HasIncoming(Gateway(g), SequenceFlow(s_1))\wedge\\ &HasOutgoing(Gateway(g), SequenceFlow(s_2))\wedge\\ &(g \in \{ExclusiveGateway, InclusiveGateway, ParallelGateway\})\wedge\\ &HasIncoming(FlowNode(a_2), SequenceFlow(s_2))\wedge\\ &((\exists e\, ConditionExpression(SequenceFlow(s_2), Expression(e)))\\ &\rightarrow GatewayPattern(a_1, a_2, g, s_2, e))\big)\end{aligned} \tag{4}$$

In our case study, the *Verify* activity serves as an instance of the input flow node (a_1) connected by an outgoing sequence flow to an exclusive gateway (g). This gateway has three output flow nodes (a_2)—the activities *Advanced Assessment*, *Simple Assessment*, and the event *Final Decision*. Routing through each sequence flow is guided by its expression (e); for instance, if the amount exceeds 5,000- the *Advanced Assessment* is triggered, and so forth.

Data Flow Dependencies Data flow dependencies illustrate how data resources, such as values, documents, and other data elements, affect and are affected by various process components like activities, events, and routing constraints. [10] conducted an extensive research about the impact of these dependencies in business processes, and identified the following types of dependencies.

Data on Data This dependency describes cases where data resources are interdependent, so that if one data resource is modified, the other is updated accordingly. This is particularly relevant when data values are functions of other data values within the business process. For example, in our case study, the *Interest* value depends on the *Amount* value. This dependency expressed in equation 5.

$$\forall r_1, \forall r_2 \left(DependsOn(DataResource(r_1), DataResource(r_2)) \right) \tag{5}$$

Data on Activity & Activity on Data The *Data on Activity* dependency exists when a data resource serves as an input to an activity, influencing its behavior or decision-making process. This is a specific case of the *InInputSet* dependency described in equation 1, where the input resource is of type *DataResource*, as shown in equation 6. For example, in our scenario, the *Amount* data resource serves as an input for either the *Advanced Assessment* or *Simple Assessment* activities. In contrast, the *Activity on Data* dependency describes instances where an activity modifies a data resource through creation, deletion, or updating. This is a specific instance of the *InOutputSet* dependency outlined in equation 2, with the output resource type as *DataResource*, as expressed in equation 7. For instance, both *Advanced Assessment* and *Simple Assessment* activities create a *Decision* data resource.

$$\forall r, \forall a \left(InInputSet(DataResource(r), Activity(a)) \right) \tag{6}$$

$$\forall r, \forall a \left(InOutputSet(DataResource(r), Activity(a)) \right) \tag{7}$$

Data on Routing Constraint This dependency exists when a data resource is used in an expression that determines the routing of a process case. In our model, this pattern is formalized in Equation 8, represented by a gateway (g) with a conditional sequence flow (s) as an output. The sequence flow (s) has an expression (e) that depends on a data resource (r). In our example, there are two instances of this pattern between the *Amount* data resource and Gateway r2. The first pattern instance is via conditional sequence flow s5, with the expression *Amount* $>$ *5000*, and the second pattern is via conditional sequence flow s6, with the expression *Amount* $<=$ *5000.*

$$\begin{aligned} \forall g, \forall s, \forall r, \forall e \big(HasOutgoing(Gateway(g), SequenceFlow(s)) \wedge \\ (s \in ConditionalSequenceFlow) \wedge \\ ConditionExpression(SequenceFlow(s), Expression(e)) \wedge \\ DependsOn(Expression(e), DataResource(r)) \\ \rightarrow DataOnRoutingConstraint(r, g, s, e) \big) \end{aligned} \tag{8}$$

Routing constraint on flow Node This dependency occurs when the activation of a flow node depends on the evaluation of a gateway and a conditional expression that relies on a data resource. In our model, this pattern is formalized in Equation 9 and is represented by a gateway (g) with a conditional sequence flow (s) as an output. The sequence flow (s) contains an expression (e) that depends on a data resource (r) and is connected to a flow node (a). In our example, this pattern exists between the *Advanced Assessment* activity and Gateway r2, via conditional sequence flow s5, with the expression *Amount > 5000.*

$$\begin{aligned}\forall g, \forall s_1, \forall s_2, \forall r, \forall e, \forall a\ \big(&HasOutgoing(Gateway(g), SequenceFlow(s_1))\wedge\\ &(s_1 \in ConditionalSequenceFlow)\wedge\\ &ConditionExpression(SequenceFlow(s_1), Expression(e))\wedge\\ &DependsOn(Expression(e), DataResource(r))\wedge\\ &HasIncoming(FlowNode(a), SequenceFlow(s_1))\\ &\rightarrow RoutingConstraintOnFlowNode(g, a, r, s_1, e)\big)\end{aligned} \tag{9}$$

3 Propagation of Availability Loss in a Business Process Model

Loss of availability propagates primarily between resources and activities, and then extends between flow nodes through control flow dependencies. If a resource functions as an input or output for an activity and becomes unavailable, the activity itself will also be unavailable. Thus, the unavailability of a resource leads to the unavailability of any dependent activities, as represented in Equation 10. For instance, if resources classified as *Assistants* become unavailable, all activities dependent on them will also be rendered unavailable. This could happen, for example, if a denial-of-service attack[1] targets their personal computers.

$$\begin{aligned}\forall r, \forall a, \forall l_1, \forall l_2\ \Big(\big(&InInputSet(Resource(r), Activity(a))\vee\\ &InOutputSet(Resource(r), Activity(a))\big)\wedge\\ \neg DataResource(r) \wedge &ParticipatesIn(Resource(r), LossOfAvailability(l_1))\\ \rightarrow ParticipatesIn(a, l_2) \wedge &LeadsTo(l_1, l_2) \wedge (l_2 \in LossOfAvailability)\Big)\end{aligned} \tag{10}$$

We distinguish between data resources and other resources, determining that a data resource must be included only in an activity's input set to affect its availability, as detailed in Equation 11. For example, if the *Amount* data resource becomes unavailable, then the activities *Advanced Assessment* and *Simple Assessment* will also be unable to execute. This scenario could occur, for instance, if an SQL injection attack[7] targets the *Amount* field in a relational database.

$$\begin{aligned}\forall r, \forall a, \forall l_1, \forall l_2\ \Big(\big(&InInputSet(DataResource(r), Activity(a))\wedge\\ &ParticipatesIn(DataResource(r), LossOfAvailability(l1))\\ \rightarrow ParticipatesIn(a, l_2) \wedge &LeadsTo(l_1, l_2) \wedge (l_2 \in LossOfAvailability)\Big)\end{aligned} \tag{11}$$

Furthermore, if data resources are interdependent, meaning a modification of one impacts another (as described in Equation 12), the unavailability of one data resource results in the unavailability of the other, as shown in Equation 12. For example, if the *Amount* data resource becomes unavailable, then the *Interest* data resource will also become unavailable as a result.

$$\begin{aligned} \forall r_1, r_2, l_1, l_2 \Big(&\big(DependsOn(DataResource(r_1), DataResource(r_2))\wedge \\ &ParticipatesIn(DataResource(r_1), LossOfAvailability(l_1))\big) \\ &\rightarrow \big(ParticipatesIn(r_2, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfAvailability})\big)\Big) \end{aligned} \tag{12}$$

Additionally, when an activity becomes unavailable, any data resources in its output set will also become unavailable, as described in Equation 13. For example, if the *Advanced Assessment* activity becomes unavailable, possibly due to the unavailability of the *Amount* data resource or the *Expert* resource, the *Decision* data resource, which depends on the assessment, will also become unavailable. This unavailability will then propagate to any activities that depend on the *Decision* data resource, leading to a cascading effect of unavailability.

$$\begin{aligned} \forall r, a, l_1, l_2 \Big(&\big(InOutputSet(DataResource(r), Activity(a))\wedge \\ &ParticipatesIn(DataResource(r), LossOfAvailability(l_1))\big) \\ &\rightarrow \big(ParticipatesIn(DataResource(r), LossOfAvailability(l_2))\wedge \\ &LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfAvailability})\big)\Big) \end{aligned} \tag{13}$$

In the case of a *NormalSequenceFlow* (see Equation 3), which represents a direct follows relationship between two flow nodes of activity or event type, the unavailability of the first flow node will prevent the execution of the subsequent flow node. This dependency is captured in Equation 14.

$$\begin{aligned} \forall a, \forall b, \forall l_1, \forall l_2 \Big(&NormalSequenceFlow(FlowNode(a), FlowNode(b))\wedge \\ &ParticipatesIn(FlowNode(a), LossOfAvailability(l_1)) \\ &\rightarrow ParticipatesIn(b, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in LossOfAvailability)\Big) \end{aligned} \tag{14}$$

In the case of a *GatewayPattern* (see Equation 4), the loss of availability propagates between two dependent flow nodes based on the type of gateway involved, this is expressed in equation 15. If the gateway is an *ExclusiveGateway*, the unavailability of an input flow node causes the output nodes, which are activated by a true evaluation of the sequence flow expression, to become unavailable. This happens because the activation of output nodes is dependent on the availability of the input node, and without it, the gateway cannot evaluate the sequence flow expression correctly, resulting in the unavailability of the outputs. If the gateway is an *InclusiveGateway*, the unavailability of all input nodes is required for the output nodes to become unavailable. This is due to the fact that the *InclusiveGateway* allows for the activation of one or more output nodes depending on the availability of the inputs; if some inputs are still available, the corresponding outputs can still be activated. However, if all inputs are unavailable, the outputs cannot be activated. If the gateway is a *ParallelGateway*, the unavailability of at least one input node leads to the unavailability of the output nodes. This is because a *ParallelGateway* requires all input nodes to be available to trigger all corresponding output nodes simultaneously, and the failure of even one input prevents the execution of the dependent flow nodes.

$$\begin{aligned}
\forall a, b, g, s_2, e, l_1, l_2 \Big(&GatewayPattern(FlowNode(a), FlowNode(b), Gateway(g),\\
&SequenceFlow(s_2), Expression(e)) \wedge\\
\big((g \in \text{ExclusiveGateway} &\wedge ParticipatesIn(FlowNode(a), LossOfAvailability(l_1)) \wedge e) \vee\\
(g \in \text{InclusiveGateway} &\wedge \forall a\, ParticipatesIn(FlowNode(a), LossOfAvailability(l_1))) \vee\\
(g \in \text{ParallelGateway} &\wedge \exists a\, ParticipatesIn(FlowNode(a), LossOfAvailability(l_1)))\big)\\
&\rightarrow \big(ParticipatesIn([b, g], l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfAvailability})\big)\Big)
\end{aligned} \tag{15}$$

In our running example, the *ExclusiveGateway* $r2$ serves as a split gateway. If the *Verify* activity becomes unavailable and the expression attached to s5 *SequenceFlow* is evaluated to be true ($Amount > 5000$), the *Advanced Assessment* activity will also become unavailable. This unavailability propagates through $r3$, a join *ExclusiveGateway*, causing it to become unavailable, which, in turn, affects $r4$ and the *Final Decision* event, making them unavailable. Furthermore, in $r5$, a split *ParallelGateway*, if the *Final Decision* event is unavailable, all subsequent activities will also become unavailable. Consequently, $r6$, a join *ParallelGateway*, propagates this unavailability to the *Process End* event. Theoretically, if $r3$ was an *InclusiveGateway*, the unavailability would not propagate to $r4$ because not all its input nodes are unavailable. In another scenario, if only the *Manager* resource experiences a loss of availability, the *Open the Credit Loan* activity will be unavailable (as stated in Equation 10). In this case, $r6$, a join *ParallelGateway*, would have two available activities and one unavailable activity as inputs. Since a *ParallelGateway* requires all input nodes to be available, the unavailability of any input node can cause a cascading effect, ultimately making the *Process End* event unavailable.

4 Propagation of Confidentiality Loss in a Business Process Model

Loss of confidentiality propagates primarily when a process entity has access to view a data resource because such access enables potential unauthorized exposure of sensitive information. One key scenario occurs when a non-data resource, such as a personal computer, participates in a loss of confidentiality event, meaning it has been compromised to the extent that data can be stolen if handled by it. In this case, if the compromised resource is part of the input set of an activity and there are data resources in the input or output sets of the same activity, the confidentiality of these data resources is also compromised. This propagation mechanism is formalized in Equation 16. For example, if the personal computer of an *Expert* is compromised, the confidentiality of the *Amount* data resource will also be compromised due to its involvement in the *Advanced Assessment* activity.

$$\begin{aligned}
&\forall r_1, r_2, a, l_1, l_2 \Big(InInputSet(Resource(r_1), Activity(a)) \wedge \\
&\neg DataResource(r_1) \wedge \neg HumanResource(r_1) \wedge \\
&ParticipatesIn(Resource(r_1), LossOfConfidentiality(l_1)) \wedge \\
&\big(InInputSet(DataResource(r_2), Activity(a)) \vee \\
&InOutputSet(DataResource(r_2), Activity(a))\big)\Big) \\
&\rightarrow \big(ParticipatesIn(r_2, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfConfidentiality})\big)
\end{aligned} \tag{16}$$

If a data resource in the input or output set of an activity experiences a loss of confidentiality, the activity's confidentiality will also be compromised, as expressed in Equation 17. This occurs because the compromise of the data directly affects the activity that relies on or produces it. For instance, if the *Amount* data resource is involved in a loss of confidentiality event, the *Advanced Assessment* activity, which depends on this resource, will also be considered compromised. This rule does not apply transitively, from activity to data resource. The propagation of confidentiality loss requires the involvement of a resource that technically exposes the information, as confidentiality cannot be compromised without such participation. Specifically, the compromise of a resource involved in the activity is necessary for this propagation to occur, as formalized in Equation 16.

$$\begin{aligned}
&\forall r, a, l_1, l_2 \Big(\big(InOutputSet(DataResource(r), Activity(a)) \vee \\
&InInputSet(DataResource(r), Activity(a))\big) \wedge \\
&ParticipatesIn(DataResource(r), LossOfConfidentiality(l_1))\Big) \\
&\rightarrow \big(ParticipatesIn(r, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfConfidentiality})\big)
\end{aligned} \tag{17}$$

5 Propagation of Integrity Loss in a Business Process Model

Loss of integrity propagates when a process entity is capable of modifying a data resource, thereby affecting the integrity of other process entities, including activities, events, and gateways. A key scenario occurs when a non-data resource, such as a personal computer, is compromised to the extent that it can alter data handled by it. If such a compromised resource is part of an activity's input set, and the activity produces data resources in its output set, the integrity of these output data resources may also be compromised, potentially altering their values as part of an attack. This propagation mechanism is formalized in Equation 18. For instance, if an *Expert*'s personal computer is compromised, the integrity of the *Decision* data resource may be affected. An attacker exploiting the compromised computer could improperly modify the *Decision* value, which is produced as the output of the *Advanced Assessment* activity.

$$\begin{aligned}\forall r_1, r_2, a, l_1, l_2 \Big(&InInputSet(Resource(r_1), Activity(a)) \wedge \\ &\neg DataResource(r_1) \wedge \neg HumanResource(r_1) \wedge \\ &ParticipatesIn(Resource(r_1), LossOfIntegrity(l_1)) \wedge \\ &InOutputSet(DataResource(r_2), Activity(a))\Big) \\ \rightarrow \big(&ParticipatesIn(r_2, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfIntegrity})\big)\end{aligned} \tag{18}$$

If data resources are interdependent—meaning that changes to one resource affect another (as described in Equation 12)—the loss of integrity in one data resource will propagate to the other, as formalized in Equation 19. For instance, if an attacker modifies the value of the *Amount* data resource, the *Interest* data resource, which depends on it, will also be altered, resulting in a loss of integrity.

$$\begin{aligned}\forall r_1, r_2, l_1, l_2 \Big(\big(&DependsOn(DataResource(r_1), DataResource(r_2)) \wedge \\ &ParticipatesIn(DataResource(r_1), LossOfIntegrity(l_1))\big) \\ \rightarrow \big(&ParticipatesIn(r_2, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfIntegrity})\big)\Big)\end{aligned} \tag{19}$$

Another case occurs when an activity depends on a compromised data resource in its input, meaning that an attacker has modified its value. In this situation, the activity's integrity will be affected. For example, if the *Amount* data resource is modified by an attacker, the integrity of activities such as *Advanced Assessment* and *Simple Assessment* will be affected because their tasks rely on the compromised values. This rule is formalized in Equation 20. Similarly, if the integrity of an activity is compromised, the integrity of its output data resource will also be impacted, as formalized in Equation 21. For instance, if the *Advanced Assessment* activity experiences a loss of integrity, the integrity of the *Decision* data resource, which is the output of that activity, will also be affected. This occurs because the integrity of the output data is directly tied to the integrity of the activity that generates it. If the activity itself produces erroneous or manipulated results due to a compromise, the resulting data will also be unreliable, leading to the propagation of integrity loss to any subsequent use of the *Decision* data resource.

$$\begin{aligned}\forall r, a, l_1, l_2 \Big(&InInputSet(DataResource(r), Activity(a)) \wedge \\ &ParticipatesIn(DataResource(r), LossOfIntegrity(l_1))\Big) \\ \rightarrow \big(&ParticipatesIn(a, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfIntegrity})\big)\end{aligned} \tag{20}$$

$$\begin{aligned}\forall r, a, l_1, l_2 \Big(&InOutputSet(DataResource(r), Activity(a)) \wedge \\ &ParticipatesIn(Activity(a), LossOfIntegrity(l_1))\Big) \\ \rightarrow \big(&ParticipatesIn(r, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfIntegrity})\big)\end{aligned} \tag{21}$$

In the scenario where data affects a routing constraint, as formalized in the *DataOnRoutingConstraint* dependency (see Equation 8), if the data participates in a loss of integrity event, the integrity of the gateway, representing the routing constraint in our model, will be impacted. This is because the gateway's role is to ensure that routing decisions are made based on valid, unaltered data. If the data is compromised, the gateway can no longer function as intended, leading to a breakdown in the routing process. This dependency is formalized in Equation 22. Similarly, if the integrity of the gateway is compromised, the integrity of its output flow nodes will also be affected. When the gateway's integrity is undermined, the flow nodes, which are dependent on the routing decisions of the gateway, will also process potentially invalid or incorrect information, thereby compromising their integrity. This dependency is formalized in Equation 23. For example, if the *Verification* data resource is compromised, neither the *Advanced Assessment* nor the *Simple Assessment* activities should be executed, as the process design relies on accurate verification to ensure proper decision-making. The *Verify* activity serves to validate the credit request before any assessments take place. If these activities are executed with a compromised verification value, they may proceed based on incorrect or invalid assumptions, leading to flawed evaluations or decisions. This deviation from the intended process compromises the integrity of the assessments, as they are based on erroneous data.

$$\begin{aligned}\forall r, g, s, e, l_1, l_2 \Big(& DataOnRoutingConstraint(DataResource(r), Gateway(g), \\ & SequenceFlow(s), Expression(e)) \wedge \\ & ParticipatesIn(DataResource(r), LossOfIntegrity(l_1)) \Big) \\ & \rightarrow \big(ParticipatesIn(g, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfIntegrity})\big)\end{aligned} \tag{22}$$

$$\begin{aligned}\forall g, a, r, s, e, l_1, l_2 \Big(& RoutingConstraintOnFlowNode(Gateway(g), FlowNode(a) \\ & DataResource(r), SequenceFlow(s), Expression(e)) \wedge \\ & ParticipatesIn(Gateway(g), LossOfIntegrity(l_1)) \Big) \\ & \rightarrow \big(ParticipatesIn(a, l_2) \wedge LeadsTo(l_1, l_2) \wedge (l_2 \in \text{LossOfIntegrity})\big)\end{aligned} \tag{23}$$

References

1. Denial of Service, Technique T0814 - ICS | MITRE ATT&CK®, `https://attack.mitre.org/techniques/T0814/`
2. Adamo, G., Ghidini, C., Di Francescomarino, C.: What is a process model composed of? a systematic literature review of meta-models in bpm. Software and Systems Modeling **20**(4), 1215–1243 (2021)
3. Annane, A., Aussenac-Gilles, N., Kamel, M.: Bbo: Bpmn 2.0 based ontology for business process representation. In: 20th European Conference on Knowledge Management (ECKM 2019). vol. 1, pp. 49–59 (2019)
4. De Leoni, M., Van Der Aalst, W.M., Van Dongen, B.F.: Data-and resource-aware conformance checking of business processes. In: Business Information Systems: 15th International Conference, BIS 2012, Vilnius, Lithuania, May 21-23, 2012. Proceedings 15. pp. 48–59. Springer (2012)
5. Goethals, F., De Backer, M., Lemahieu, W., Snoeck, M., Vandenbulcke, J., Infrastructures, S.l.E.E.: Identifying dependencies in business processes. In: Communication and Coordination in Business Processes (LAP-CCBP) Workshop, Kiruna, Sweden, June. vol. 22 (2005)

6. Neubauer, T., Klemen, M., Biffl, S.: Secure business process management: a roadmap. In: First International Conference on Availability, Reliability and Security (ARES'06). pp. 8–pp. IEEE (2006)
7. OWASP Foundation: SQL Injection - OWASP (2024), `https://owasp.org/www-community/attacks/SQL_Injection`, accessed: 2024-11-14
8. Pashchenko, I., Vu, D.L., Massacci, F.: A qualitative study of dependency management and its security implications. In: Proceedings of the 2020 ACM SIGSAC conference on computer and communications security. pp. 1513–1531 (2020)
9. Sales, T.P., Baião, F., Guizzardi, G., Almeida, J.P.A., Guarino, N., Mylopoulos, J.: The common ontology of value and risk. In: Conceptual Modeling: 37th International Conference, ER 2018, Xi'an, China, October 22–25, 2018, Proceedings 37. pp. 121–135. Springer (2018)
10. Tsoury, A., Soffer, P., Reinhartz-Berger, I.: Data impact analysis in business processes: Automatic support and practical implications. Business & Information Systems Engineering **62**, 41–60 (2020)
11. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer, Cham (2012). `https://doi.org/10.1007/978-3-642-28616-2`