Secure software placement and configuration[☆]

Zoltán Ádám Mann

University of Duisburg-Essen, Essen, Germany

Abstract

Finding the best way to place the components of an application on a set of heterogeneous servers is a challenging task, especially if some components are associated with security requirements. To address security requirements, several security controls may be available, some of them software-based (e.g., encryption), others hardware-based (e.g., trusted execution environments). Security controls may incur widely varying performance overhead. There is a non-trivial interplay between application placement (which component to place on which server) and the configuration of security controls (which security control to activate for which component). On the one hand, placing a component on a secure server may make it unnecessary to use software-based security controls for the component. On the other hand, the overhead of using a specific security control may increase the resource requirements of a component so that it does not fit onto its designated server. Therefore, this paper addresses the joint problem of application placement and configuration of security controls. We formalize the problem and use mixed integer quadratic programming to solve it. A case study is used to demonstrate that the proposed approach can automatically determine the placement and configuration of complex applications.

Keywords: application placement, security control, risk management, data protection, cloud computing, fog computing, edge computing

1. Introduction

The target infrastructure for the deployment of applications is increasingly complex, heterogeneous, and not fully known when the application is designed [1, 2]. For example, applications can be designed so that they can be deployed to a cloud-based infrastructure, which may be a private cloud, one of the many public cloud offerings, or a combination of these [3, 4, 5]. Different instances of the same application may be run on different infrastructures and even the same instance of the application may be re-deployed to a changed infrastructure during its lifetime. These opportunities, which are made possible by modern computing technologies like virtualization and microservices and methodologies like DevOps, have led to increased flexibility and agility in software provisioning [6]. In the recently introduced fog computing paradigm [7], applications may be placed on a continuum of resources with widely varying capabilities, from cloud data centers through fog nodes to end devices [8, 9]. This makes it possible to place applications so as to optimize several important system metrics, like latency, resource utilization, and energy consumption [10].

On the other hand, not knowing the exact details of the infrastructure on which the application will be placed is a challenge for software design. One of the key issues in a cloud deployment is security [11], because both the providers and other tenants of cloud services may be able to attack applications and data in the cloud [12]. Different infrastructures may be associated with significantly different security properties [13, 14]. Hence, applications must be designed in such a way that they incorporate multiple security techniques, from which the techniques to be activated can be chosen during deployment or during run time when the characteristics of the infrastructure are known [15, 16, 17]. For example, if an application is placed in a public cloud, it may store, transfer, and process sensitive data only in encrypted form, which might incur a considerable performance overhead. If the same application is placed on a secure infrastructure, encryption can be deactivated, thus avoiding the performance overhead.

Different kinds of security controls may be available for securing an application on an infrastructure:

- Hardware-based security controls, e.g., a server in a trusted private cloud or a server that has been made invisible to potential attackers [18] can be assumed to be secure.
- Software-based security controls, e.g., encryption. This can also include advanced techniques using encrypted data, for instance searchable encryption [19].
- Combinations of hardware-based and software-based security controls. E.g., a server may offer secure hardware enclaves – memory regions protected against access from unauthorized processes [20] – which applications with appropriate logic may leverage to protect sensitive data.

In this paper, we consider the placement of an application consisting of a set of components, which are connected by con-

[☆]This paper was published in Future Generation Computer Systems, volume 110, pages 243-253, 2020

nectors. Some of the components and connectors may store, process, or transmit sensitive data and hence must be protected. As part of the deployment process, the components and connectors have to be placed on servers and links between servers, respectively. Moreover, the security controls have to be configured, i.e., it has to be decided which ones to activate.

There is a non-trivial interplay between application placement and the configuration of security controls. On one hand, the placement of a component or connector determines which hardware-based security controls can be used to protect the component or connector. On the other hand, the overhead of using a specific security control may increase the resource requirements of a component or connector so that it does not fit onto its designated server or link.

Because of the interdependency between placement and configuration, manual deployment is a daunting task even for moderately complex applications [21]. Existing tools fall short of handling the complicated constraints stemming from the different kinds of security controls, their dependence on the placement and their impact on performance. Therefore, this paper makes the following contributions:

- We formalize the joint problem of application placement and configuration of security controls, taking into account hardware-based, software-based, and combined hardware-software-based security controls, as well as their performance overhead.
- We devise an algorithm for creating a deployment satisfying all security and capacity constraints, by transforming the problem to a mixed integer quadratic program.
- A case study is used to demonstrate that the proposed approach can automatically determine the placement and configuration of complex applications. Moreover, controlled experiments are used to investigate the scalability of the proposed approach.

To the best of our knowledge, this paper is the first to investigate the interplay between application placement and the configuration of security controls, and to propose an algorithm for this joint problem.

2. Problem model

Figure 1 gives an overview about the used problem model. The details are explained below.

2.1. Problem inputs

The application to deploy is given as a set of components, as well as connectors that connect components. Thus, the application is modeled as an undirected graph $G_{\mathrm{SW}} = (V_{\mathrm{SW}}, E_{\mathrm{SW}})$, where the vertices in V_{SW} are the components and the edges in E_{SW} are the connectors (see also Table 1). Components have a given CPU requirement (modeled as `Component.cpuReq` in Figure 1) and connectors have a given bandwidth requirement (`Connector.bwReq`). Moreover, each component and connector may be declared as sensitive (`Component.sensitive` and `Connector.sensitive`), which means that it has to be protected by some security control.

Table 1: Notation overview

Symbol	Meaning
V_{SW}	Set of application components
E_{SW}	Set of connectors between the components
V_{HW}	Set of servers
E_{HW}	Set of links between the servers
C_{Comp}	Set of all component security controls
C_{Conn}	Set of all connector security controls
C_{Serv}	Set of all server security controls
C_{Link}	Set of all link security controls
C_{all}	Set of all security controls
$C(x)$	Set of security controls provided by x
$p_{\mathrm{V}}(x)$	Server that should host component x
$p_{\mathrm{E}}(x)$	Link that should host connector x
$a(z)$	True iff security control z should be activated
$s(x, p_{\mathrm{V}}, a)$	Size of component x, given p_{V} and a
$s(x, p_{\mathrm{E}}, a)$	Size of connector x, given p_{E} and a

The available hardware infrastructure consists of a set of servers as well as links connecting the servers. Thus, the hardware is modeled as an undirected graph $G_{\mathrm{HW}} = (V_{\mathrm{HW}}, E_{\mathrm{HW}})$, where the vertices in V_{HW} are the servers and the edges in E_{HW} are the links. It should be noted that the links represent end-to-end communication links between the servers. The way of mapping these end-to-end links on paths of the physical network is beyond the scope of this paper. Servers have a given CPU capacity (modeled as `Server.cpuCap`) and links have a given bandwidth (`Link.bwCap`).

Each server and link may provide some hardware-based security controls (server security control respectively link security control). Similarly, each component and connector may provide some software-based security controls (component security control respectively connector security control). A security control is either always on (`alwaysOn=true`), or otherwise (i.e., if `alwaysOn=false`) it can be turned on and off as modeled with the `activated` attribute. Moreover, each security control is associated with an increment, modeling the overhead incurred when the given security control is turned on. Specifically:

- `ComponentSecurityControl.cpuReqIncrement` is the amount by which the CPU requirement of the component associated with the security control increases if the security control is activated.
- `ConnectorSecurityControl.bwReqIncrement` is the amount by which the bandwidth requirement of the connector associated with the security control increases if the security control is activated.
- `ServerSecurityControl.cpuReqIncrement` is the amount by which the CPU requirement of all components hosted on the server associated with the security control increases if the security control is activated.

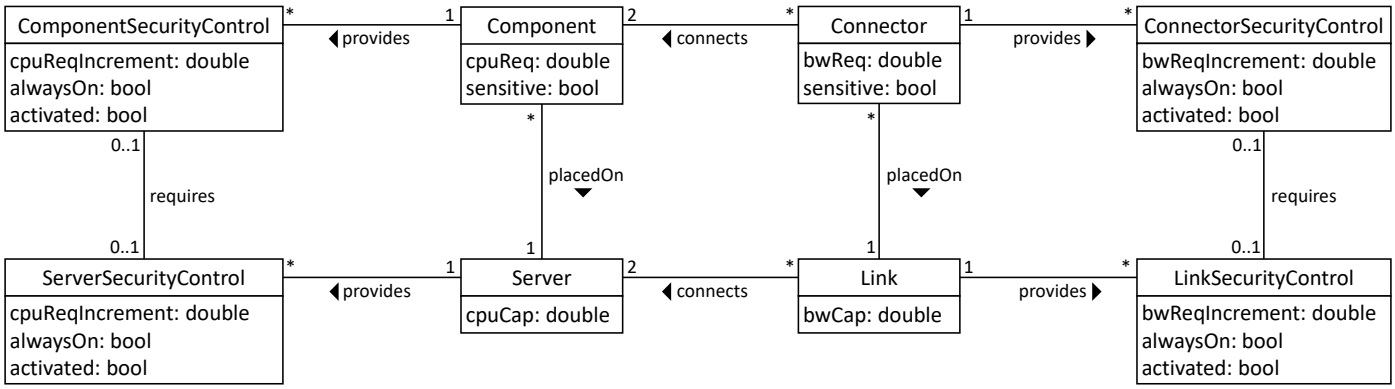

Figure 1: Problem model (using UML class diagram notation)

- `LinkSecurityControl.bwReqIncrement` is the amount by which the bandwidth requirement of all connectors mapped on the link associated with the security control increases if the security control is activated.

A security control may work on its own, but there can also be pairs of security controls that only work in tandem. Specifically, it is possible that a `ComponentSecurityControl` and a `ServerSecurityControl` can only work together (modeled by the bi-directional relation labeled `requires` between them), or similarly, that a `ConnectorSecurityControl` and a `LinkSecurityControl` can only work together. The `requires` relations are bi-directional and define that both of the involved security controls must be available and active to achieve security. For example, a server that provides secure hardware enclaves as `ServerSecurityControl` and a component that contains the necessary logic to exploit such enclaves as `ComponentSecurityControl` can be used in combination to achieve security, but neither of them would be sufficient without the other.

It is important to note the cardinalities on the `provides` relations. For example, the cardinalities on the `provides` relation between `Component` and `ComponentSecurityControl` mean that a component may provide any number (zero or more) of component security controls, whereas a component security control belongs to exactly one component. In other words, each component has its own component security controls, independently from the other components. As a consequence, the component security controls of individual components can be activated or deactivated independently from each other.

The set of all component security controls is denoted by C_{Comp}, the set of all connector security controls by C_{Conn}, the set of all server security controls by C_{Serv} and the set of all link security controls by C_{Link}. $C(x)$ denotes the set of security controls provided by x (if x is a component, then $C(x)$ consists of component security controls; if x is a connector, then $C(x)$ consists of connector security controls etc.). $C_{\text{all}} = C_{\text{Comp}} \cup C_{\text{Conn}} \cup C_{\text{Serv}} \cup C_{\text{Link}}$ is the set of all security controls.

2.2. Problem outputs

The task is to determine a placement of the application and the configuration of the security controls. The placement is defined by two functions $p_{\text{V}} : V_{\text{SW}} \to V_{\text{HW}}$ (mapping each component to a server) and $p_{\text{E}} : E_{\text{SW}} \to E_{\text{HW}}$ (mapping each connector to a link). The configuration of the security controls is a function $a : C_{\text{all}} \to \{\text{true}, \text{false}\}$, specifying which security controls should be activated and which ones not. Thus, the output of the problem is the tuple $(p_{\text{V}}, p_{\text{E}}, a)$.

The actual size (i.e., CPU requirement) of a component depends on a, since each activated security control of the given component may increase the size of the component. Moreover, the actual size of a component also depends on p_{V}: if the component is mapped on a server with a server security control activated, this also may increase the size of the component. Thus, the actual size of component x, given mapping p_{V} and configuration a, is:

$$s(x, p_{\text{V}}, a) = x.\texttt{cpuReq} + \sum_{z \in C(x),\ a(z)=\text{true}} z.\texttt{cpuReqIncrement} + \sum_{z \in C(p_{\text{V}}(x)),\ a(z)=\text{true}} z.\texttt{cpuReqIncrement} \quad (1)$$

Similarly, the actual size (i.e., bandwidth requirement) of a connector x is calculated by:

$$s(x, p_{\text{E}}, a) = x.\texttt{bwReq} + \sum_{z \in C(x),\ a(z)=\text{true}} z.\texttt{bwReqIncrement} + \sum_{z \in C(p_{\text{E}}(x)),\ a(z)=\text{true}} z.\texttt{bwReqIncrement} \quad (2)$$

2.3. Constraints

For the placement to be valid, the following constraints must hold[1]:

$$vw \in E_{\text{SW}} \quad \Rightarrow \quad p_{\text{E}}(vw) \text{ connects } p_{\text{V}}(v) \text{ and } p_{\text{V}}(w) \quad (3)$$

$$\forall y \in V_{\text{HW}} : \sum_{x \in V_{\text{SW}},\ p_{\text{V}}(x)=y} s(x, p_{\text{V}}, a) \leq y.\texttt{cpuCap} \quad (4)$$

$$\forall y \in E_{\text{HW}} : \sum_{x \in E_{\text{SW}},\ p_{\text{E}}(x)=y} s(x, p_{\text{E}}, a) \leq y.\texttt{bwCap} \quad (5)$$

[1] To keep notations simple, we use vw to refer to the edge connecting the vertices v and w

Equation (3) ensures that the placement of components and the placement of connectors are compatible, i.e., a connector is placed on a link that connects the two servers which host the components connected by the connector. Equation (4) is the capacity constraint for servers: for each server y, the total size of components placed on y must not exceed the capacity of y. Note that the size of the components depends on the placement and on the configuration of security controls, as expressed by (1). Equation (5) is the analogous capacity constraint for links.

There is a corner case that requires special attention. Two components v_1 and v_2 connected by a connector v_1v_2 may be placed on the same server. In order to be able to define $p_{\mathrm{E}}(v_1v_2)$ in such cases, we assume that each server is connected to itself by a link of infinite bandwidth, i.e.

$$\forall u \in V_{\mathrm{HW}} : \; e = uu \in E_{\mathrm{HW}}, \; e.\texttt{bwCap} = \infty. \tag{6}$$

For the configuration of security controls, an obvious constraint is that security controls that are always enabled are indeed active:

$$\forall z \in C_{\mathrm{all}} : \; z.\texttt{alwaysOn} \Rightarrow a(z) = \mathrm{true}. \tag{7}$$

Further, it must be ensured that each sensitive component is protected somehow. This is expressed by the following constraint:

$$\begin{aligned} &\forall x \in V_{\mathrm{SW}} : \; x.\mathrm{sensitive} \Rightarrow \\ &\Big[(\exists z \in C(x), a(z), z' = z.\texttt{requires} \in C(p_{\mathrm{V}}(x)), a(z')) \\ &\qquad \vee (\exists z \in C(x), a(z), z.\texttt{requires} = \texttt{null}) \\ &\qquad \vee (\exists z' \in C(p_{\mathrm{V}}(x)), a(z'), z'.\texttt{requires} = \texttt{null}) \Big] \end{aligned} \tag{8}$$

Equation (8) stipulates that each sensitive component x must be secured in one of the following ways:

- By a component security control z of the component, which is activated (i.e., $a(z)$ is true). Furthermore, this component security control requires
  - either a server security control z' which is provided by the server on which x is placed ($z' = z.\texttt{requires} \in C(p_{\mathrm{V}}(x))$) and z' is also activated ($a(z')$ is true).
  - or no server security control (that is, $z.\texttt{requires} = \texttt{null}$);
- Alternatively, the component x is protected by a server security control z', provided by the server on which x is placed (i.e., $z' \in C(p_{\mathrm{V}}(x))$). Furthermore, this server security control is activated ($a(z')$ is true) and requires no component security control ($z'.\texttt{requires} = \texttt{null}$).

It should also be noted that constraint (8) contains both the $a()$ and $p_{\mathrm{V}}()$ functions, making some of the interplay between placement of components and configuration of security controls explicit.

Similarly to constraint (8), we have the following constraint for securing sensitive connectors:

$$\begin{aligned} &\forall x \in E_{\mathrm{SW}} : \; x.\mathrm{sensitive} \Rightarrow \\ &\Big[(\exists z \in C(x), a(z), z' = z.\texttt{requires} \in C(p_{\mathrm{E}}(x)), a(z')) \\ &\qquad \vee (\exists z \in C(x), a(z), z.\texttt{requires} = \texttt{null}) \\ &\qquad \vee (\exists z' \in C(p_{\mathrm{E}}(x)), a(z'), z'.\texttt{requires} = \texttt{null}) \Big] \end{aligned} \tag{9}$$

Altogether, the problem that we are addressing consists of finding the functions $(p_{\mathrm{V}}, p_{\mathrm{E}}, a)$ such that constraints (3)-(5) and (7)-(9) are satisfied. We term this problem the Joint Placement and Configuration Problem (JPCP).

It should be noted that an instance of the JPCP may have 0, 1, or multiple solutions. If there are multiple solutions, an objective function could be used to determine the "best" solution. This is a possible future extension (see also Section 7 for a further discussion). In the presented problem formulation, any solution that satisfies all constraints is considered equally appropriate.

3. Proposed approach

The core of the proposed approach is the conversion of the JPCP to a Mixed Integer (quadratically constrained) Quadratic Programming (MIQP) formulation. This is described first, followed by the details of the actual solution procedure.

3.1. MIQP formulation

Mixed Integer (quadratically constrained) Quadratic Programming involves formulating the problem using a finite set of – real-valued or integer-valued – variables, and a finite set of constraints, which are equations or inequalities comprised of linear or quadratic expressions of the variables. It is more common to use Mixed Integer Linear Programming, in which the constraints are all linear, to solve application placement problems [22, 23], but as we will see, we need the higher expressive power of quadratic constraints to express the more complex relationships in connection with the security controls.

The variables of the MIQP formulation, denoted by Greek letters, are summarized in Table 2. The placement function $p_{\mathrm{V}}()$ is encoded using a set of binary variables: $\xi_{u,v}$, where $u \in V_{\mathrm{SW}}$ and $v \in V_{\mathrm{HW}}$. Similarly, $p_{\mathrm{E}}()$ is encoded using another set of binary variables: $\eta_{e,l}$, where $e \in E_{\mathrm{SW}}$ and $l \in E_{\mathrm{HW}}$. The activation function $a()$ is encoded via a further set of binary variables: ζ_z, where $z \in C_{\mathrm{all}}$. Finally, the auxiliary function $s()$ is encoded using a set of real-valued variables: σ_x, where $x \in V_{\mathrm{SW}} \cup E_{\mathrm{SW}}$.

The variables must fulfill several constraints. First, it has to be ensured that each component is placed on exactly one server and each connector is placed on exactly one link:

$$\forall u \in V_{\mathrm{SW}} : \; \sum_{v \in V_{\mathrm{HW}}} \xi_{u,v} = 1 \tag{10}$$

$$\forall e \in E_{\mathrm{SW}} : \; \sum_{l \in E_{\mathrm{HW}}} \eta_{e,l} = 1 \tag{11}$$

Table 2: Variables

Name	Domain	Description
$\xi_{u,v}$	$\{0,1\}$	1 iff component u is placed on server v
$\eta_{e,l}$	$\{0,1\}$	1 iff connector e is placed on link l
ζ_z	$\{0,1\}$	1 iff security control z is active
σ_x	$\mathbb{R}_+$	Size of component or connector x

Analogously to Equation (3), the placement of components and the placement of connectors must be compatible:

$$\forall u_1u_2 \in E_{\text{SW}}, v_1v_2 \in E_{\text{HW}}: \quad \eta_{u_1u_2,v_1v_2} \leq \xi_{u_1,v_1} \cdot \xi_{u_2,v_2} + \xi_{u_1,v_2} \cdot \xi_{u_2,v_1} \tag{12}$$

Note that this constraint is quadratic. It ensures that, if connector u_1u_2 is placed on link v_1v_2 (i.e., the left-hand side is 1), then either $\xi_{u_1,v_1} \cdot \xi_{u_2,v_2} = 1$ (i.e., component u_1 is placed on server v_1 and component u_2 is placed on server v_2), or $\xi_{u_1,v_2} \cdot \xi_{u_2,v_1} = 1$ (i.e., u_1 is placed on v_2 and u_2 is placed on v_1).

Analogously to Equation (1), it must be ensured that the size of the components is computed correctly in the σ_x variables, based on the activation of the related security controls as defined by the ζ_z variables and on the placement of the components as defined by the $\xi_{x,y}$ variables:

$$\begin{aligned}\forall x \in V_{\text{SW}}: \ \sigma_x = x.\texttt{cpuReq} \\ + \sum_{z \in C(x)} \zeta_z \cdot z.\texttt{cpuReqIncrement} \\ + \sum_{y \in V_{\text{HW}}} \sum_{z \in C(y)} \xi_{x,y} \cdot \zeta_z \cdot z.\texttt{cpuReqIncrement}\end{aligned} \tag{13}$$

Note that this equation is also quadratic. Similarly, in analogy to Equation (2), the size of connectors has to be computed:

$$\begin{aligned}\forall x \in E_{\text{SW}}: \ \sigma_x = x.\texttt{bwReq} \\ + \sum_{z \in C(x)} \zeta_z \cdot z.\texttt{bwReqIncrement} \\ + \sum_{y \in E_{\text{HW}}} \sum_{z \in C(y)} \eta_{x,y} \cdot \zeta_z \cdot z.\texttt{bwReqIncrement}\end{aligned} \tag{14}$$

Analogously to Equations (4)-(5), the capacity constraints must be enforced for each server and each link. Two constraints (both are quadratic) are needed as follows:

$$\forall v \in V_{\text{HW}}: \sum_{u \in V_{\text{SW}}} \xi_{u,v} \cdot \sigma_u \leq v.\texttt{cpuCap} \tag{15}$$

$$\forall l \in E_{\text{HW}}: \sum_{e \in E_{\text{SW}}} \eta_{e,l} \cdot \sigma_e \leq l.\texttt{bwCap} \tag{16}$$

Analogously to Equation (7), it has to be ensured that each security control that is specified to be always on will really be active:

$$\forall z \in C_{\text{all}}, \ z.\texttt{alwaysOn}: \ \zeta_z = 1. \tag{17}$$

Finally, it has to be ensured analogously to Equations (8) and (9) that each sensitive component and connector is protected somehow. First the constraint for components:

$$\begin{aligned}\forall u \in V_{\text{SW}}, \ u.\text{sensitive}: \sum_{\substack{z \in C(u) \\ z.\texttt{requires=null}}} \zeta_z + \\ + \sum_{\substack{z \in C(u) \\ z.\texttt{requires}=z' \neq \texttt{null}}} \zeta_z \cdot \zeta_{z'} \cdot \sum_{\substack{v \in V_{\text{HW}} \\ z' \in C(v)}} \xi_{u,v} + \\ + \sum_{v \in V_{\text{HW}}} \sum_{\substack{z'' \in C(v) \\ z''.\texttt{requires=null}}} \xi_{u,v} \cdot \zeta_{z''} \geq 1\end{aligned} \tag{18}$$

The first term corresponds to the situation that the component u is secured by one of its own security controls z, which does not require a server security control ($z.\texttt{requires} = \texttt{null}$) and is activated ($\zeta_z = 1$). In the second term, z does require a server security control z' which is also activated ($\zeta_{z'} = 1$), and the component u is placed on a server v offering the security control z'. The third term describes the situation in which u is placed on a server v which is secured by a server security control z'' that is activated and does not require any component security control as counterpart.

Constraint (18) is cubic and not quadratic. By introducing auxiliary binary variables $\omega_{z,z'}$, we can replace Constraint (18) by the following pair of quadratic constraints:

$$\forall z \in C_{\text{Comp}}, z' \in C_{\text{Serv}}: \quad \omega_{z,z'} = \zeta_z \cdot \zeta_{z'} \tag{19}$$

$$\begin{aligned}\forall u \in V_{\text{SW}}, \ u.\text{sensitive}: \sum_{\substack{z \in C(u) \\ z.\texttt{requires=null}}} \zeta_z + \\ + \sum_{\substack{z \in C(u) \\ z.\texttt{requires}=z' \neq \texttt{null}}} \omega_{z,z'} \cdot \sum_{\substack{v \in V_{\text{HW}} \\ z' \in C(v)}} \xi_{u,v} + \\ + \sum_{v \in V_{\text{HW}}} \sum_{\substack{z'' \in C(v) \\ z''.\texttt{requires=null}}} \xi_{u,v} \cdot \zeta_{z''} \geq 1\end{aligned} \tag{20}$$

Similarly, the following constraints ensure that each sensitive connector is secured:

$$\forall z \in C_{\text{Conn}}, z' \in C_{\text{Link}}: \quad \omega_{z,z'} = \zeta_z \cdot \zeta_{z'} \tag{21}$$

$$\begin{aligned}\forall e \in E_{\text{SW}}, \ e.\text{sensitive}: \sum_{\substack{z \in C(e) \\ z.\texttt{requires=null}}} \zeta_z + \\ + \sum_{\substack{z \in C(e) \\ z.\texttt{requires}=z' \neq \texttt{null}}} \omega_{z,z'} \cdot \sum_{\substack{l \in E_{\text{HW}} \\ z' \in C(l)}} \eta_{e,l} + \\ + \sum_{l \in E_{\text{HW}}} \sum_{\substack{z'' \in C(l) \\ z''.\texttt{requires=null}}} \eta_{e,l} \cdot \zeta_{z''} \geq 1\end{aligned} \tag{22}$$

Altogether, the JPCP is equivalent to finding an assignment for the variables in Table 2 and the auxiliary binary variables $\omega_{z,z'}$, which fulfills constraints (10)-(17) and (19)-(22).

Algorithm 1 Solving the JPCP

1: Furnish each server with a (fictive) link to itself
2: Transform JPCP into MIQP
3: Solve MIQP
4: Transform result of MIQP to a solution of JPCP
5: (Optional) Deactivate unnecessary security controls

3.2. Solution procedure

The steps of the proposed algorithm are summarized in Algorithm 1, and explained in the following.

First, each server is furnished with a fictive link with infinite capacity that connects the server with itself, in line with Equation (6). Moreover, since these new links do not represent real vulnerabilities[2], they should be regarded as unconditionally protected. For this reason, each new link is associated with a (fictive) link security control that is always on, requires no connector security control, and has 0 overhead.

In the next two steps, the mixed integer quadratic program is created and solved using an appropriate solver. For this purpose, we use the Gurobi Optimizer[3], a popular mathematic programming solver. Hence, the MIQP is created and solved using the API provided by Gurobi.

In the fourth step, the result of Gurobi is processed. There are two possibilities for the result. The first possibility is that the MIQP is not solvable, which means that the original JPCP is also not solvable. This can happen for example if the capacity of the infrastructure is not sufficient or if the available security controls are insufficient to secure every sensitive component and connector. If the problem is unsolvable, our algorithm outputs this information, so that software developers or system administrators can implement appropriate changes in the application or in the infrastructure to enable a secure placement. The other possibility is that the solver returns an assignment to the variables that satisfies all constraints. From this, the solution of the JPCP can be decoded:

- The values of the ζ_z variables directly specify which security controls to activate.
- For the $\xi_{u,v}$ variables, constraint (10) ensures that there is exactly one $v \in V_{\mathrm{HW}}$ for each $u \in V_{\mathrm{SW}}$ for which $\xi_{u,v} = 1$. This v is the server on which component u is to be placed. The placement of connectors arises from the $\eta_{e,l}$ variables in an analogous way.

After this step, a solution of the JPCP has been determined (if one exists), so the solution procedure could be finished. However, for practical reasons, we propose to add a further post-processing step, in which unnecessary security controls are deactivated. The reason for this step is that the solution delivered by the solver may define the activation of security controls that are not strictly necessary. For example, consider a single sensitive component with a CPU requirement of 5 units and a single server with a CPU capacity of 10 units. Assume that the component can be protected either by a component security control or by a server security control, both of which add an overhead of 1 unit to the CPU requirement of the component. In this case, the JPCP has three solutions, as shown in Table 3. All three solutions satisfy each constraint (in particular, the capacity of the server is not overloaded and the sensitive component is protected in each case), and are hence each valid. The solver may return any one of these solutions, since each of them is a correct solution of the JPCP. However, the third solution can be considered less practical than the others, because it uses two security controls although one would be sufficient, leading to a waste of resources.

Table 3: Activation of the security controls in different solutions in the example

solution nr.	component security control	server security control
1	active	not active
2	not active	active
3	active	active

Algorithm 2 Deactivating unnecessary security controls

1: **for all** $z \in C_{\mathrm{all}}$ **do**
2: **if** z.activated = true **and** z is unnecessary **then**
3: z.activated $\leftarrow$ false
4: **end if**
5: **end for**

We formalize this phenomenon with the following notion:

Definition 1. *In a solution of the JPCP, an active security control z is called* unnecessary*, if after deactivating z, still all constraints of the JPCP are satisfied.*

Deactivating unnecessary security controls does not influence the correctness of the solution, but helps to avoid the unnecessary waste of resources. Therefore, we post-process the solution by iterating through all activated security controls and if some could be deactivated without violating the constraints, then we deactivate them. This procedure is shown in Algorithm 2.

4. Case study

We implemented our approach in the form of a Java program[4], using the Gurobi Optimizer as an external solver.

To demonstrate the applicability of our approach and illustrate its operation, we applied it to the cloud-based variant of the CoCoME case study [24]. CoCoME models cloud services that support the typical trading operations of a supermarket chain, like the management of stores, inventory management, and product dispatching. As such, CoCoME offers a realistic case study covering computationally intensive application

[2]Note that the server may be vulnerable. However, this must be addressed in the context of the placement of the components on the server. If two components exchanging sensitive information – which makes the components sensitive as well – are mapped to this server, then the components or the server must offer appropriate security controls. Thus, the information exchange between the two components does not lead to further risks.

[3]http://www.gurobi.com

[4]The implementation is publicly available from `https://sourceforge.net/p/vm-alloc/sspc`.

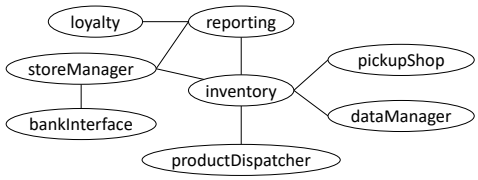

Figure 2: Example application architecture

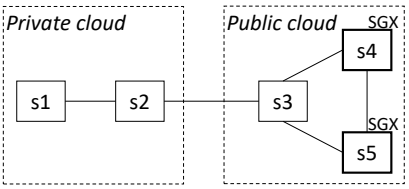

Figure 3: Example infrastructure

components, data transfers among the components, and security concerns.

The architecture of the CoCoME application in terms of components and connectors is shown in Figure 2. In our case study, we assume that all components and all connectors are sensitive. This is based on the fact that the data processed by the components and exchanged along the connectors give insight into the trade secrets associated with the operation of the supermarket chain, such as logistics details and pricing policies. If the application were running without protection on a public infrastructure, a competitor could gain access to this information. In addition, several components and connectors could also leak personal information about customers and employees which need to be protected according to the applicable data protection regulation.

The following security controls are considered:

- Hardware-based: some servers are in a private cloud, behind the corporate firewall. These servers, as well as the links among them, are assumed to be secure.
- Software-based: some components can be protected by using an encrypted database like CryptDB for storing data. Using CryptDB, Popa et al. observed overhead in the range of 14.5%-26.0% [25], hence we also use overhead values in this range.
- Software-based: connectors can be protected by using encryption. Using AES-256 encryption, Pawar et al. reported bandwidth overhead in the range of 13.8%–51.4% depending on the used communication technology [26], hence we also use overhead values in this range.
- Hardware- and software-based: some servers offer secure hardware enclaves, which some components can leverage. Specifically, we assume the use of Intel SGX (Software Guard Extensions[5]) enclaves, provided by modern Intel processors. When using SGX enclaves, Arnautov et al. experienced CPU overhead in the range of 16.4%–62.9%, depending on application type [27].

Figure 3 depicts the target infrastructure. Servers s1 and s2 as well as the link between them are in the private cloud and thus considered secure. Servers s4 and s5 offer SGX enclaves, so that application components that support SGX can leverage them for protection. Server s3 offers no protection. Also, all links other than the one inside the private cloud are not protected.

[5]https://software.intel.com/en-us/sgx

Table 4: Component security controls

Name	Security control	Dependency	Overhead
storeManager			
bankInterface			
loyalty	sgx-use	sgx	40%
—"—	encryption		20%
reporting	encryption		20%
inventory	encryption		30%
pickupShop	sgx-use	sgx	40%
dataManager	sgx-use	sgx	50%
productDispatcher	sgx-use	sgx	60%

The security controls offered by the components are shown in Table 4. The storeManager and bankInterface components offer no software-based security control, while the loyalty component offers two possible security controls, and the others offer one. The security control "sgx-use" depends on the hardware-based security control "sgx". Table 5 shows the software-based security controls for the connectors. Each connector offers the possibility to use encryption, albeit with different overhead values. None of the component or connector security controls is marked as "always on".

For the sake of simplicity, we assume that each component and each connector has the same base size, namely 10 units. Moreover, each server has the same capacity κ_{server} and each link has the same capacity κ_{link}.

To demonstrate the way the problem is modeled, Figure 4 depicts a part of the JPCP instance corresponding to the case study, as an instantiation of the general model shown in Figure 1. In particular, it shows the reporting and loyalty components and the connector between them, as well as all the component and connector security controls provided by these. In addition, the servers s1 and s4 are shown, as well as the link between servers s1 and s2, together with the server and link

Table 5: Connector security controls

Connector	Security control	Overhead
storeManager ↔ bankInterface	encryption	30%
storeManager ↔ reporting	encryption	10%
storeManager ↔ inventory	encryption	20%
loyalty ↔ reporting	encryption	20%
reporting ↔ inventory	encryption	30%
pickupShop ↔ inventory	encryption	20%
inventory ↔ dataManager	encryption	10%
inventory ↔ productDispatcher	encryption	10%

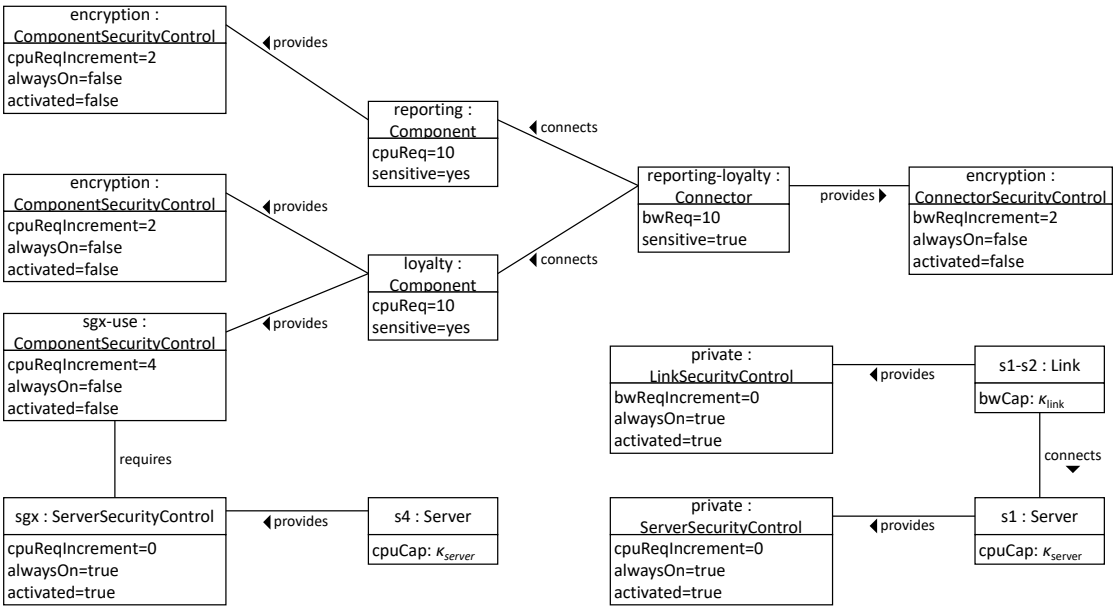

Figure 4: Excerpt of the JPCP instance corresponding to the case study (using UML object diagram notation)

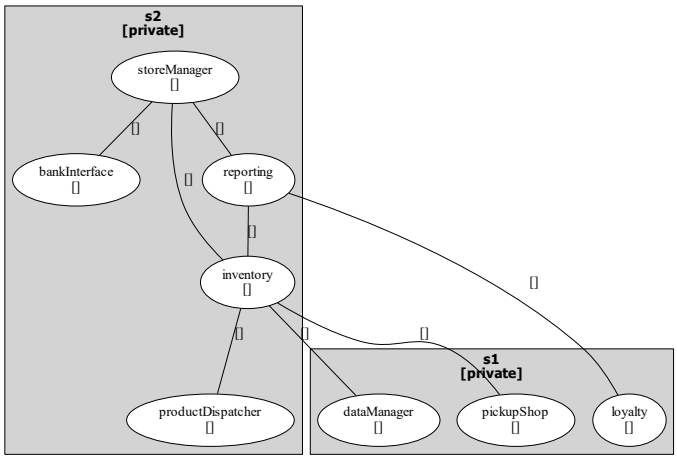

Figure 5: Results for $\kappa_{\text{server}} = 60$ and $\kappa_{\text{link}} = 40$

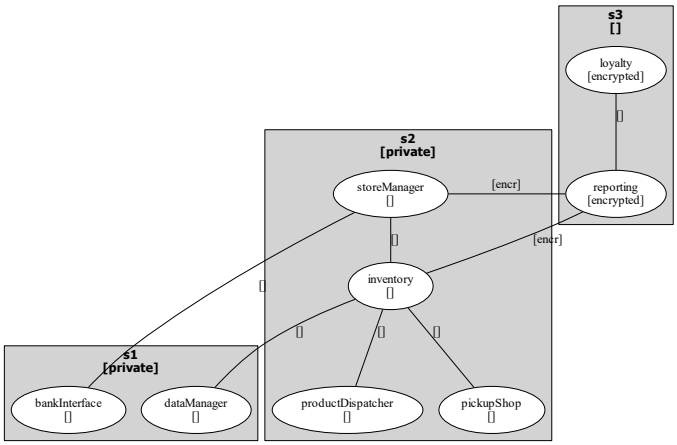

Figure 6: Results for $\kappa_{\text{server}} = 49$ and $\kappa_{\text{link}} = 37$

security controls provided by these.

Figure 5 shows the deployment created by our program for $\kappa_{\text{server}} = 60$ and $\kappa_{\text{link}} = 40$. In this and the next figures, the gray boxes represent the servers, with their names in boldface, followed by their security controls in brackets. The links are not shown explicitly to avoid clutter. The ovals are the components, containing their names, and in brackets the *activated* security controls. The edges between the ovals are the connectors. Here, too, only the activated security controls are given in brackets. As Figure 5 shows, the given settings are quite loose, so that the whole application can be placed on the two servers in the private cloud, requiring no further security controls to be activated.

Figure 6 shows the deployment created for the slightly tighter capacities of $\kappa_{\text{server}} = 49$ and $\kappa_{\text{link}} = 37$. (These and the following numbers for the parameters κ_{server} and κ_{link} were carefully selected for demonstration purposes, such that each new pair of parameter values leads to a new deployment, and each interesting case is covered.) In this case, the resources of the private cloud are not sufficient anymore, so that some components have to be spilled to server s3 in the public cloud. The components and connectors within the private cloud still need no software-based security controls. However, encryption is automatically activated for the two components placed on s3 as well as for the two connectors running between s2 and s3 (for connectors, encryption is abbreviated as "encr"). This is necessary since neither server s3 nor the link between s2 and s3 offers any hardware-based security control.

Further decreasing the server and link capacities to $\kappa_{\text{server}} = 39$ and $\kappa_{\text{link}} = 35$, we obtain the deployment shown in Figure 7. Clearly, the two previous deployments would not be appropriate for these capacities. It can be checked that the new deployment, which makes heavy use of the servers in the public cloud, satisfies all constraints. In particular, each component that is placed on a server in the public cloud is protected either by encryption

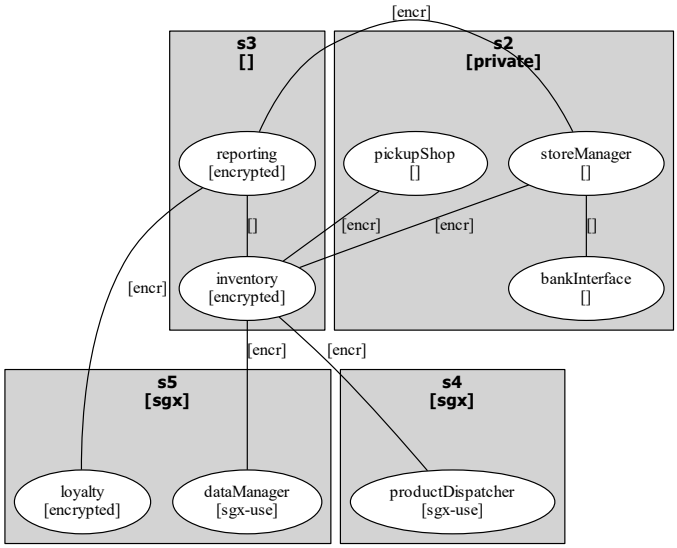

Figure 7: Results for $\kappa_{\text{server}} = 39$ and $\kappa_{\text{link}} = 35$

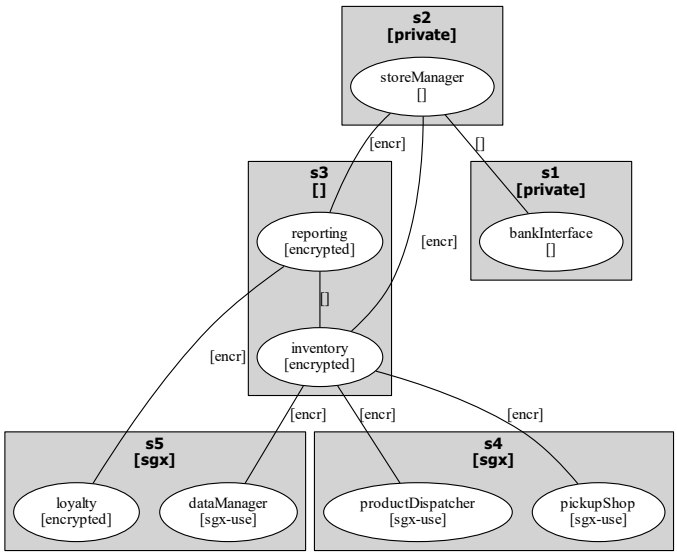

Figure 8: Results for $\kappa_{\text{server}} = 35$ and $\kappa_{\text{link}} = 30$

or by SGX enclaves. The "sgx-use" security control is used in conjunction with SGX-capable servers.

With an even stricter setup of $\kappa_{\text{server}} = 35$ and $\kappa_{\text{link}} = 30$, the previous deployment would not be valid anymore because the link between servers s2 and s3 would be overloaded by the three connectors, due to the overhead incurred by their encryption. For this setup, we obtain a deployment that uses all five available servers, as shown in Figure 8. This deployment is more balanced, leading to at most two connectors for each link, thus fitting the lower capacity of the links.

Obviously, after some point, a valid deployment becomes impossible. For example for $\kappa_{\text{server}} = 28$ and $\kappa_{\text{link}} = 24$, the program's answer is that there is no valid deployment.

The case study has shown how complex the interdependencies between the placement of the components and the configuration of security controls can become even for applications of modest size. For human experts, finding a correct deployment in such cases may be very challenging. In contrast, our approach automatically delivers a correct deployment (or determines that this is not possible).

Table 6: Test data for scalability experiments

Parameter	Value
Server CPU capacity	50
Server security controls	50% unconditional, 50% SGX
Link bandwidth (BW)	40
Link security controls	50% unconditional
Hardware topology	random tree
Sensitive components	100%
Component CPU requirement	10
Component security controls	100% SGX-capable, can be turned on/off, penalty=5
Sensitive connectors	100%
Connector BW requirement	10
Connector security controls	100% supports encryption, can be turned on/off, penalty=2
Software topology	random tree

5. Scalability

While the proposed approach is guaranteed to always deliver correct results, and the case study has shown the fundamental applicability of the approach to the deployment of realistic applications, scalability might still be a barrier to the practical application of our approach. In particular, there are n^m possibilities to map m software components on n servers. Additionally, if each software component provides one security control that can be turned on or off, this leads to 2^m possible configurations for the activation of the security controls, resulting in a search space of size $n^m \cdot 2^m$.

In practice, the solver can be more efficient than this theoretical bound. To investigate the scalability of our approach, we performed a number of controlled experiments[6] with synthetically generated test problems of different size.

We generate n servers with CPU capacity 50 (see also Table 6). Each server has either – with probability 0.5 – an unconditional hardware-based security control (the server is in a trusted environment, e.g., the private cloud), or supports SGX, which can be exploited by appropriate software components. The servers are connected in a random tree topology. Each link has bandwidth 40. Each link has, with probability 0.5, an unconditional hardware-based security control. We generate m software components; each is considered sensitive and each has a CPU requirement of 10. Moreover, each component is assumed to be able to use SGX. This security control can be turned on or off. When it is activated, it increases the CPU requirement of the component by 5. The components are also connected in a random tree topology. Each connector has a

[6]The measurements were performed on a Lenovo ThinkPad X1 laptop with Intel Core i5-4210U CPU @ 1.70GHz and 8GB RAM. The Gurobi Optimizer was executed with a timeout of 300 seconds.

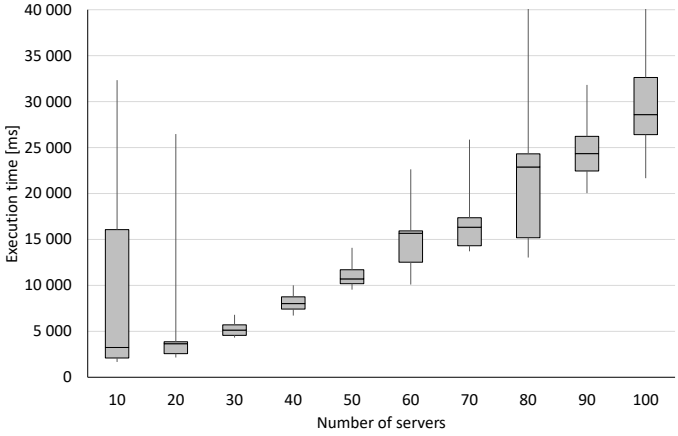

Figure 9: Execution time for increasing number of servers, while the number of components is constant

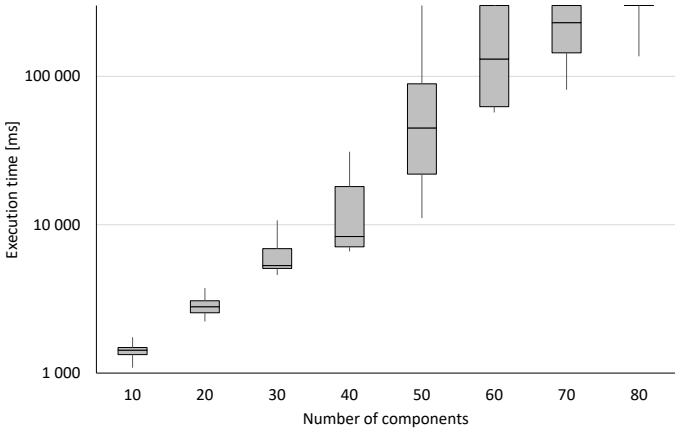

Figure 10: Execution time for increasing number of components, while the number of servers is constant

bandwidth requirement of 10, is sensitive, and can be protected by encryption. The encryption of a connector can be switched on or off. When it is activated, it increases the bandwidth requirement of the connector by 2.

Figure 9 shows the execution time for an increasing number n of servers, while the number of components is fixed at $m = 30$. The number of servers increases from 10 to 100 in increments of 10. For each data point, we performed 10 measurements. The boxes in the figure go from the 1st to the 3rd quartile of the results, with the mean marked with a horizontal line in the box, and the whiskers going to the minimum respectively maximum of the results. For few servers (especially for $n = 10$, to some extent also for $n = 20$), the mean execution time is low, but the variance of the execution time is quite high. This is because these problem instances are small, but can sometimes still be challenging, since the infrastructure has hardly enough capacity to host all the software components. For a higher number of servers, the mean execution time scales roughly linearly with n. Even for 100 servers, the mean execution time is below 30 seconds.

Figure 10 shows the execution time for a constant number of $n = 30$ servers, with the number of components increasing from 10 to 80 with increments of 10. The execution time grows rapidly with an increasing number of components (note the logarithmic scale on the vertical axis). Until about 50 components, execution is quite fast (for 50 components, the median execution time is 44.9 seconds), but afterwards the solver is stopped increasingly often by the 300-seconds timeout.

In conclusion, the proposed algorithm is reasonably fast for applications with a limited number of components (up to 50 components), even for large infrastructures. If the number of components is higher, scalability becomes a problem. Thus our algorithm is appropriate for coarse-grained decompositions of applications. Algorithmic improvements to cope with a higher number of components could be a goal for future research. For example, the proposed method could be used in a divide-and-conquer approach to determine the placement and configuration of parts of a large-scale application.

6. Related work

To the best of our knowledge, the problem formulation and the solution approach presented in this paper are novel. Nevertheless, there are several papers on automated application deployment taking into account security requirements that are related to our work.

Existing approaches explored different ways of capturing and enforcing security requirements during software deployment. A simple way of representing security requirements of application components and security capabilities of infrastructure resources is by using security levels. Goettelmann et al. used this approach for specifying security constraints, and then applied a combination of a greedy algorithm and tabu search for optimizing the deployment [28]. Wen et al. also used a similar security model and a custom heuristic algorithm for deployment optimization [29, 30]. Mezni et al. also adopted a similar security model and used particle swarm optimization to find a good deployment [31].

Our security model can also be cast in this terminology. We use two security levels for the security requirements of application components and connectors (sensitive vs. not sensitive) and two security levels for the security capabilities of the resources (protected vs. not protected). In practice, using a more fine-grained classification is difficult as it is hard to obtain realistic values for the security capabilities of resources especially in public clouds. Indeed, estimating the trustworthiness of resources is a research topic on its own [32, 33], which is orthogonal to our work. On the other hand, our security model is more general in the sense that it also allows the dynamic activation of security controls. Moreover, as opposed to the algorithms proposed by Goettelmann et al., Wen et al., and Mezni et al., our algorithm is guaranteed to always find a secure deployment whenever such a deployment exists.

Instead of security levels, a more precise way of capturing security constraints is by defining the specific security controls required by the different application components and connectors, respectively offered by the different hardware resources. Massonet et al. used this approach for specifying security constraints [34]. They proposed a method based on constraint pro-

gramming that finds an optimized deployment plan respecting the given security requirements. Forti et al. also used a similar approach, extended with probabilities and trust relations among stakeholders [35]. The model underlying these approaches is limited to security controls offered by the infrastructure, whereas our model can also capture the possible activation of security controls offered by the application itself.

In our earlier work, we devised custom heuristics to find a deployment, taking into account a specific security control, namely secure hardware enclaves [36]. However, that approach is only a heuristic, whereas the approach presented here is guaranteed to always find a suitable deployment, whenever such a deployment exists. Also, the presented approach supports a much wider range of security controls.

A different kind of security constraint, taken into account in some existing papers, is relating to the colocation of specific application components on the same cloud resource. Since colocation in a multi-tenant system allows side-channel attacks, limiting colocation can enhance security. Fdhila et al. considered such constraints when partitioning and deploying composite applications to federated clouds [37]. Agarwal and Duong also focused on the risks of colocation in public infrastructure clouds [38]. Our approach could be easily extended with such constraints, which is an opportunity for further research.

Tang et al. considered the service selection problem with the aim of minimizing the risk of privacy violations [39]. While that problem also bears some resemblance to ours, there are significant differences. In particular, the approach of Tang et al. does not consider security controls. Moreover, that approach is a heuristic, with no guarantees.

Also other problems related to automated software deployment have been addressed. In particular, several authors investigated the problem of scheduling a workflow using the resources of federated or hybrid clouds [40, 41]. Another related area is the allocation of massively parallel tasks using cloud resources [42, 43]. However, these papers focus only on costs and performance or execution time, without considering security requirements. In contrast, our approach also guarantees the fulfillment of security requirements. Moreover, in the above works the communication structure between tasks is either constrained to be acyclic, which is an unrealistic assumption for many applications, or not considered at all. In contrast, our approach works with arbitrary communication topologies among the components of an application.

Workflow scheduling was considered in conjunction with data protection concerns by Wen et al. [44]. However, in that work, data protection constraints are limited to the specification of the allowed set of data centers for a task. Our approach is much more general as it also supports the activation of security controls. Data protection was also investigated by Amato et al., but from a monitoring point of view [45], which is orthogonal to our work.

In fog computing, security and privacy concerns also play an important role [46]. When placing applications on a fog infrastructure, several approaches take security and privacy concerns into account by means of constraining the placement of certain application modules to trusted hosts [47, 2, 8]. In contrast, our approach supports a more sophisticated handling of security and privacy constraints, by specifying that a software component can only be placed on an otherwise insecure node if the software component protects itself through appropriate software-based security controls.

7. Discussion

The aim of this section is to discuss some of the limitations of the presented approach and possible future directions to address those limitations.

Our problem model handles one type of interaction between security controls: that two security controls must be used together to achieve a security goal. However, there can also be other types of interactions between security controls. E.g., a security control may be effective only in conjunction with at least one from a set of other security controls. As a future research direction, the use of Boolean formulae could be investigated as a means of specifying which combinations of security controls can be considered acceptable.

Another issue not covered by the present paper is related to the numbers in the problem model (CPU requirements of components, CPU capacity of servers, bandwidth requirements of connectors etc.). It is assumed by the proposed approach that these numbers can be obtained as input. This is in line with the assumptions of other related approaches [29, 31, 34]. The numbers may stem from several sources, such as from estimation by system designers based on their previous experience, from theoretical analysis, or from measurements (benchmarking, profiling). Moreover, they may represent typical-case or worst-case behavior, depending on system priorities and requirements. The way how these numbers are determined is an important field of research on its own, which is orthogonal to the work presented in this paper.

As already mentioned, the presented formulation of JPCP is not an optimization problem. If there are several solutions satisfying all constraints, they are considered to be equally appropriate. As future work, different optimization objectives could be introduced. The optimization objective depends on what is important in the specific application area, and may include financial costs, energy consumption, availability, performance etc. As long as only one optimization objective is concerned, the proposed approach can be extended easily. If more than one optimization objective is to be considered, this may require the use of multi-criteria optimization techniques.

8. Conclusions

In this paper, we addressed the problem of joint placement and configuration of applications during their deployment, focusing on security requirements. We have argued that on the one hand, the activation of software-based security controls leads to overhead that impacts the placement possibilities, and on the other hand, the placement of the components influences the availability of hardware-based security controls for the application components and thus also the need for software-based

security controls. Because of these interdependencies, the joint consideration of component placement and configuration of security controls is an important, but also highly complex problem. We formalized this problem and proposed an algorithm for it, which is based on the conversion of the problem to a mixed integer quadratic program. The practical applicability of the suggested approach was demonstrated by applying it to the deployment of the CoCoME application to a hybrid cloud. Experimenting with different server and link capacities, it was shown how our approach automatically finds different configurations which satisfy all consistency, capacity, and security requirements. Controlled experiments with problem instances of different size showed that the execution time of our algorithm is moderately influenced by the number of servers, but heavily influenced by the number of components. In its current form, the algorithm is reasonably fast for applications with about 50 components.

There are several promising directions for future research. Conceptually, it would be interesting to extend the presented approach with further types of requirements (e.g., anti-colocation constraints) or with optimization objectives (e.g., minimizing costs). Algorithmically, improving the performance of the algorithm – especially for applications with many components – is an important goal. Practically, we plan to integrate the presented approach into established deployment toolchains, for instance using TOSCA (Topology and Orchestration Specification for Cloud Applications) as a standardized format for describing deployments [48, 49].

Acknowledgment

This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grants 731678 (RestAssured) and 871525 (FogProtect).

References

[1] A. Dearie, Software deployment, past, present and future, in: Future of Software Engineering (FOSE), IEEE, 2007, pp. 269–284.

[2] M. Nardelli, V. Cardellini, V. Grassi, F. L. Presti, Efficient operator placement for distributed data stream processing applications, IEEE Transactions on Parallel and Distributed Systems 30 (8) (2019) 1753–1767.

[3] N. Ferry, A. Rossini, F. Chauvel, B. Morin, A. Solberg, Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems, in: IEEE 6th International Conference on Cloud Computing (CLOUD), IEEE, 2013, pp. 887–894.

[4] R. Moreno-Vozmediano, R. S. Montero, I. M. Llorente, IaaS cloud architecture: From virtualized datacenters to federated cloud infrastructures, Computer 45 (12) (2012) 65–72.

[5] Z. A. Mann, Modeling the virtual machine allocation problem, in: Proceedings of the International Conference on Mathematical Methods, Mathematical Models and Simulation in Science and Engineering, 2015, pp. 102–106.

[6] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices architecture enables DevOps: Migration to a cloud-native architecture, IEEE Software 33 (3) (2016) 42–52.

[7] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture 98 (2019) 289–330.

[8] A. Brogi, S. Forti, A. Ibrahim, Predictive analysis to support fog application deployment, Fog and edge computing: Principles and paradigms (2019) 191–222.

[9] A. Brogi, S. Forti, C. Guerrero, I. Lera, How to place your apps in the fog: State of the art and open challenges, Software: Practice and Experience (2019) https://doi.org/10.1002/spe.2766.

[10] Z. Á. Mann, Optimization problems in fog and edge computing, in: Fog and Edge Computing: Principles and Paradigms, Wiley, 2019, pp. 103–121.

[11] F. Dong, J. Shen, Q. He, Advances in cloud computing and big data analytics, Concurrency and Computation: Practice and Experience 30 (20) (2018) e4960.

[12] C. A. Ardagna, R. Asal, E. Damiani, Q. H. Vu, From security to assurance in the cloud: A survey, ACM Computing Surveys 48 (1) (2015) 2:1–2:50.

[13] D. Zissis, D. Lekkas, Addressing cloud computing security issues, Future Generation Computer Systems 28 (3) (2012) 583–592.

[14] T. Basso, R. Moraes, N. Antunes, M. Vieira, W. Santos, W. Meira Jr, PRIVAaaS: privacy approach for a distributed cloud-based data analytics platforms, in: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE Press, 2017, pp. 1108–1116.

[15] M. Anisetti, C. Ardagna, E. Damiani, F. Gaudenzi, A semi-automatic and trustworthy scheme for continuous cloud service certification, IEEE Transactions on Services Computing (2017) DOI: 10.1109/TSC.2017.2657505.

[16] S. Calo, E. Lupu, E. Bertino, S. Arunkumar, G. Cirincione, B. Rivera, A. Cullen, Research challenges in dynamic policy-based autonomous security, in: IEEE International Conference on Big Data, IEEE, 2017, pp. 2970–2973.

[17] S. Schoenen, Z. Á. Mann, A. Metzger, Using risk patterns to identify violations of data protection policies in cloud systems, in: Service-Oriented Computing – ICSOC 2017 Workshops, Springer, 2017, pp. 296–307.

[18] J. Park, J. Noh, M. Kim, B. B. Kang, Invi-server: Reducing the attack surfaces by making protected server invisible on networks, Computers & Security 67 (2017) 89–106.

[19] F. Han, J. Qin, J. Hu, Secure searches in the cloud: A survey, Future Generation Computer Systems 62 (2016) 66–75.

[20] V. Costan, I. A. Lebedev, S. Devadas, Sanctum: Minimal hardware extensions for strong software isolation, in: USENIX Security Symposium, 2016, pp. 857–874.

[21] M. Eisa, M. Younas, K. Basu, H. Zhu, Trends and directions in cloud service selection, in: IEEE Symposium on Service-Oriented System Engineering (SOSE), IEEE, 2016, pp. 423–432.

[22] T. Verbelen, T. Stevens, F. De Turck, B. Dhoedt, Graph partitioning algorithms for optimizing software deployment in mobile cloud computing, Future Generation Computer Systems 29 (2) (2013) 451–459.

[23] D. Bartók, Z. A. Mann, A branch-and-bound approach to virtual machine placement, in: Proceedings of the 3rd HPI Cloud Symposium "Operating the Cloud", 2015, pp. 49–63.

[24] R. Heinrich, K. Rostami, R. Reussner, The CoCoME platform for collaborative empirical research on information system evolution, Tech. rep., Karlsruhe Reports in Informatics (2016).

[25] R. A. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan, CryptDB: processing queries on an encrypted database, Communications of the ACM 55 (9) (2012) 103–111.

[26] P. S. Pawar, A. Sajjad, T. Dimitrakos, D. W. Chadwick, Security-as-a-service in multi-cloud and federated cloud environments, in: IFIP International Conference on Trust Management, Springer, 2015, pp. 251–261.

[27] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, C. Fetzer, SCONE: Secure Linux containers with intel SGX, in: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16), 2016, pp. 689–703.

[28] E. Goettelmann, W. Fdhila, C. Godart, Partitioning and cloud deployment of composite web services under security constraints, in: IEEE International Conference on Cloud Engineering (IC2E), IEEE, 2013, pp. 193–200.

[29] Z. Wen, J. Cala, P. Watson, A scalable method for partitioning workflows with security requirements over federated clouds, in: IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2014, pp. 122–129.

[30] Z. Wen, J. Cała, P. Watson, A. Romanovsky, Cost effective, reliable and secure workflow deployment over federated clouds, IEEE Transactions on Services Computing 10 (6) (2017) 929–941.
[31] H. Mezni, M. Sellami, J. Kouki, Security-aware SaaS placement using swarm intelligence, Journal of Software: Evolution and Process 30 (8) (2018) e1932.
[32] M. Al-khafajiy, T. Baker, M. Asim, Z. Guo, R. Ranjan, A. Longo, D. Puthal, M. Taylor, COMITMENT: A fog computing trust management approach, Journal of Parallel and Distributed Computing 137 (2020) 1–16.
[33] E. Alemneh, S.-M. Senouci, P. Brunet, T. Tegegne, A two-way trust management system for fog computing, Future Generation Computer Systems 106 (2020) 206–220.
[34] P. Massonet, J. Luna, A. Pannetrat, R. Trapero, Idea: Optimising multi-cloud deployments with security controls as constraints, in: International Symposium on Engineering Secure Software and Systems, Springer, 2015, pp. 102–110.
[35] S. Forti, G.-L. Ferrari, A. Brogi, Secure cloud-edge deployments, with trust, Future Generation Computer Systems 102 (2020) 775–788.
[36] Z. Á. Mann, A. Metzger, Optimized cloud deployment of multi-tenant software considering data protection concerns, in: 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), IEEE, 2017, pp. 609–618.
[37] W. Fdhila, M. Dumas, C. Godart, L. García-Bañuelos, Heuristics for composite web service decentralization, Software & Systems Modeling 13 (2) (2014) 599–619.
[38] A. Agarwal, T. N. B. Duong, Secure virtual machine placement in cloud data centers, Future Generation Computer Systems 100 (2019) 210–222.
[39] M. Tang, S. Zeng, J. Liu, B. Cao, Service selection based on user privacy risk evaluation, in: International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage, Springer, 2017, pp. 308–320.
[40] E. N. Alkhanak, S. P. Lee, R. Rezaei, R. M. Parizi, Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues, Journal of Systems and Software 113 (2016) 1–26.
[41] J. Zhu, X. Li, R. Ruiz, X. Xu, Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources, IEEE Transactions on Parallel and Distributed Systems 29 (6) (2018) 1401–1415.
[42] W.-J. Wang, Y.-S. Chang, W.-T. Lo, Y.-K. Lee, Adaptive scheduling for parallel tasks with QoS satisfaction for hybrid cloud environments, The Journal of Supercomputing 66 (2) (2013) 783–811.
[43] M. Malawski, K. Figiela, J. Nabrzyski, Cost minimization for computational applications on hybrid cloud infrastructures, Future Generation Computer Systems 29 (7) (2013) 1786–1794.
[44] Y. Wen, J. Liu, W. Dou, X. Xu, B. Cao, J. Chen, Scheduling workflows with privacy protection constraints for big data applications on cloud, Future Generation Computer Systems (2018) in press.
[45] F. Amato, L. Coppolino, S. D'Antonio, N. Mazzocca, F. Moscato, L. Sgaglione, An abstract reasoning architecture for privacy policies monitoring, Future Generation Computer Systems 106 (2020) 393–400.
[46] A. Alrawais, A. Alhothaily, C. Hu, X. Cheng, Fog computing for the internet of things: Security and privacy issues, IEEE Internet Computing 21 (2) (2017) 34–42.
[47] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, F. Desprez, Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed IoT applications in the fog, in: Proceedings of the 33rd Annual ACM Symposium on Applied Computing, ACM, 2018, pp. 751–760.
[48] OASIS, Topology and Orchestration Specification for Cloud Applications (TOSCA), version 1.0., OASIS Standard, http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html (2013).
[49] J. Bellendorf, Z. A. Mann, Specification of cloud topologies and orchestration using TOSCA: a survey, Computing (2019) https://doi.org/10.1007/s00607–019–00750–3.