Abstract

Building large systems by composing reusable services is not a new idea, it is at least 25 years old. Still, only recently the scenario of dynamic interchangeable services that are consumed via public networks is becoming reality. Following the Software as a Service (SaaS) paradigm, an increasing number of complex applications is offered as a service that themselves can be used composed for building even larger and more complex applications. This will lead to situations in which users are likely to unknowingly consume services in a dynamic and ad hoc manner.

Leaving the rather static (and mostly on-premise) service composition scenarios of the past 25 years behind us, dynamic service compositions, have not only the potential to transform the software industry from a business perspective, they also requires new approaches for addressing the security, trustworthiness needs of users. The EU FP7 project Aniketos develops new technology, methods, tools and security services that support the design-time creation and run-time dynamic behaviour of dynamic service compositions, addressing service developers, service providers and service end users.

In this talk, we will motivate several security and trustworthiness requirements that occur in dynamic service compositions and discuss the solutions developed within the project Aniketos. Based on our experiences, we will discuss open research challenges and potential opportunities for potential opportunities for applying type systems.

The Aniketos Project

Enable composite services to establish and maintain security and trustworthiness

Goals of the Aniketos platform:
- Design-time discovery, composition and evaluation, threat awareness
- Runtime adaptation or change in service configuration
- Runtime monitoring, detection, notification

Two related dimensions:
- Trustworthiness: Reputation, perception, centralised vs. distributed
- Security properties: Behaviour, contracts, interfaces, formal verification

About Me

My Employer: SAP AG
- Vendor of enterprise software systems
- World’s third largest software vendor
- More than 25 industries
- 63% of the world’s transaction revenue touches an SAP system
- 64,422 employees worldwide

Personal Background:
- Senior Researcher: Security, Formal Methods, Software Engineering
- Security Expert: supporting all phases of a SDLC
Outline

1. (Secure) Service Composition: Past, Present, and Future
2. The Aniketos Approach: Overview
3. The Aniketos Approach: Exemplary Deep Dive
4. Service Compositions: A Curse or Blessing for Security?

The Past: Service Compositions

"Service: a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface ... with constraints and policies as specified by the service description."

OASIS Reference Model for Service Oriented Architecture

- At least 20 years old:
  - RPCs introduced in 1980s
  - CORBA published in 1991
- Motivated by
  - re-useability
  - reliability
- Used within organisations
- Frameworks considered to be heavy-weight

The Past: Secure Service Compositions

Recall the past:
- networks were expensive (and slow)
- only a few people had access to networked systems

Security model:
- non-technical trust
  - small numbers of users allowing a personal relationship
  - system operators trust their users
- security perimeter
  - limited access
  - firewalls
  - controlled system access

The Present: Service Composition

Motivated by business needs:
- cost-savings
- flexibility
- Used across organisations

Environment
- fast networks
- many users
- relatively static compositions
**The Present: Secure Service Composition**

**Access Control**

**Goal:**
- Control access to services, resources (data), . . .

**The core:**
- Usually:
  - users, roles, access rights, . . .
- In special cases:
  - data labelling or information flow

**On top:**
- Separation of Duty
- Binding of Duty
- Delegation

---

**Photo:** Syohei Arai

---

**The Present: Secure Service Composition**

**Protecting Data (and Physical Goods)**

**Goal:**
- Ensure
  - confidentiality
  - integrity (safety)
  - of data (and goods)

**The core:**
- Need-to-Know
- Fingerprints
- Encryption
- Sensors

---

**Photo:** Bundesarchiv, Bild 183-R0117-0003 / CC-BY-SA

---

**The Present: Secure Service Composition**

**Compliance and Additional Requirements**

Many regulated markets
- Basel II/III, SoX, PCI
- HIPAA

Many customer-specific regulations
- Own governance to mitigate risks
- Own business code of conduct
- Fraud detection/prevention
- Non-observability

Customers are individually audited
- No “one certificate fits all” solution

Security should not hinder business

---

**Photo:** Ralf Roletschek

---

**The Future: Service Composition**

- Software as a service
  - complex components
  - updates controlled by provider

- Many external services
- No central orchestrator
- Complex data flows

---

**Photo:** Ralf Roletschek
The Future: Secure Service Composition

We need to ensure the already discussed requirements.

Many additional challenges, e.g.,
- Customer
  - ensure compliance in a changing environment
- Developer
  - provide secure, scalable services
- Provider
  - provide secure offerings
  - protect own infrastructure
  - protect data of customers

Outline
1. (Secure) Service Composition: Past, Present, and Future
2. The Aniketos Approach: Overview
3. The Aniketos Approach: Exemplary Deep Dive
4. Service Compositions: A Curse or Blessing for Security?

The Aniketos Process

Modeling Composition Plans using BPMN

- Human-centric tasks
- Automated tasks (services)
- Orchestration of services
- Start/end states
- Logical control flow (if/and/or)
- Error states
Secure Implementation of Atomic Services

Check the compliance of atomic service implementations.

- **Human tasks:**
  - Define the user interface (e.g., HTML, Java)
  - Create, read, or update of process variables

- **Service tasks:**
  - Define the business logic (e.g., Java, Web service specific configuration)
  - Create, read, or update of process variables
Implementing “Contact Travel Service Company”

```java
package corp.acme;
public class SendOrderToTravelAgency implements JavaDelegate {
    @Override
    public void execute(DelegateExecution ex) throws Exception {
        String lastname = (String) ex.getVariable("user_lastname");
        String firstname = (String) ex.getVariable("user_firstname");
        String email = (String) ex.getVariable("user_email");
        String reason = (String) ex.getVariable("travel_business_reason");
        String destination = (String) ex.getVariable("travel_destination");
        String duration = (String) ex.getVariable("travel_duration");

        // Code for accessing the web service
        QName SERVICE_NAME = new QName("http://travel.corp/", "TravelService");
        URL WSDLURL = new URL("http://travel.corp/TravelService/Service.asmx?WSDL");
        Service travelService = new Service(WSDLURL, SERVICE_NAME);
        ServiceSoap port = travelService.getServiceSoap();

        // send order to travel service
        port.orderTravelAssistance(firstname, lastname, email, reason, destination, duration);
    }
}
```

The Problem: RBAC with Separation of Duty

Role-based access control (RBAC)
- Subjects are assigned to roles
- Permissions assign roles to tasks (resources)

Separation of duty (SoD)
- restrict subjects in executing tasks

We analyse:
- Does the RBAC configuration comply to the SoD requirements?
  - yes: static SoD
  - no: dynamic SoD
- In case of a compliance violation:
  - change RBAC configuration
  - ensure dynamic enforcement of SoD
Analysing (Dynamic | Static) Separation of Duty

Does the access control enforce a separation of duty constraint

- Translate the composition plan to ASLan
  
  \( \text{hc } \text{rbac} \_ \text{ac}(\text{Subject}, \text{Role}, \text{Task}) := \text{CanDoAction}(\text{Subject}, \text{Role}, \text{Task}) \)
  
  \( \text{hc } \text{poto(T6)} := \text{poto(Staff, Request Travel)} \)
  
  \( \text{hc } \text{poto(T6)} := \text{poto(Manager, Approve Absence)} \)
  
  \( \text{hc } \text{poto(T7)} := \text{poto(Manager, Approve Budget)} \)

- Specify the test goal
  
  \( \text{attack} \_ \text{state } \text{sod} \_ \text{securitySod1} \_ \text{1}(\text{Subject0}, \text{Subject1}, \text{Instance1}, \text{Instance2}) := \text{executed}(\text{Subject0}, \text{task(Request Travel, Instance1)}). \)
  
  \( \text{executed}(\text{Subject1}, \text{task(Approve Budget, Instance2)}).
  
  \text{executed}(\text{Subject3}, \text{task(Approve Absence, Instance3)}) \)
  
  \( \text{\&not}(\text{equal}(\text{Subject0}, \text{Subject1})) \)
  
  \( \text{\&not}(\text{equal}(\text{Subject1}, \text{Subject2})) \)
  
  \( \text{\&not}(\text{equal}(\text{Subject2}, \text{Subject3})) \)

- Run the model checker
- Translate the analysis result back to BPMN (visualisation)

Outline

1. (Secure) Service Composition: Past, Present, and Future
2. The Aniketos Approach: Overview
3. The Aniketos Approach: Exemplary Deep Dive
4. Service Compositions: A Curse or Blessing for Security?

Why Are Enterprises Moving Services to the Cloud

The shift to the cloud (SaaS) is driven by economical considerations

- total cost of ownership
- need for adaptability (flexibility)
- ...

Core assumption: specialised cloud providers

- operate systems cheaper (licenses, upgrades, etc.)
- achieve higher reliability
- provide more flexibility (elasticity, features, etc.)
- ...

And security?
Security Chances and Risks of Service Compositions

Chances
- regular updates
  - XSS, SQL injection, etc.
- system administration
  - misconfiguration
- secured data centres
  - specialises systems
- ...

Challenges
- how to trust the provider
  - data disclosure
- new attacks
  - tenants as attackers
  - providers as attackers
- how to control delegation
  - subcontracting
- ...

Challenges for Type-based Approaches (in Industry)
- Real systems are not build from scratch
  - existing frameworks
  - legacy systems
  - ...
- Developers hate to write type annotations
  - BufferedReader in = new BufferedReader(converter);
  - we need to advertise
    - type inference,
    - a concise syntax for typed languages, and
    - helpful error messages
- Weakly/dynamically typed languages are gaining popularity
  - light-weight typed based analysis approaches
  - type systems for
    - well-defined subsets that
    - can interact with the whole language (libraries, etc.)

Conclusion
- The interesting challenges are still ahead of us!
- Real systems are large and complex:
  - many programming languages or frameworks
  - many security technologies
  - highly distributed
- There is a trend towards weakly typed languages
  - can we provide type-based analysis for such systems
  - can we provide (strongly) typed alternatives that
    - provide similar flexibility
    - can integrate existing frameworks
- Security is more than CIA
  - needs to be ensured on all levels
    - implementation level vulnerabilities
    - configuration errors
    - security frameworks/implementation (authentication, crypto)
  - business-level compliance
  - legal frameworks

Thank you for your attention!
Any questions or remarks?
References

Achim D. Brucker and Isabelle Hang.
Secure and compliant implementation of business process-driven systems.

Achim D. Brucker, Isabelle Hang, Gero Lückemeyer, and Raj Ruparel.
SecureBPMN: Modeling and enforcing access control requirements in business processes.

Achim D. Brucker, Francesco Malmignati, Madjid Merabti, Qi Shi, and Bo Zhou.
A framework for secure service composition.

Luca Compagna, Pierre Guillemot, and Achim D. Brucker.
Business process compliance via security validation as a service.