Eine zustandsbehaftete Web Service Firewall für BPEL

SPRING2 2007
Dortmund

Nils Gruschka, Meiko Jensen, Norbert Luttenberger

Arbeitsgruppe Kommunikationssysteme
Institut für Informatik
Christian-Albrechts-Universität zu Kiel
Overview

- BPEL Workflow
  - Introduction
  - Example
- Attacks on BPEL
- Countermeasure
  - Firewall for BPEL engines
  - BPEL state automaton
  - Automaton generation
- Conclusion
Introduction

- **Workflow**
  - sequential and parallel execution of *communication* activities
  - representing operational aspects of business processes

- **Web Services**
  - widespread communication paradigm
  - suitable for workflow communication

- **Business Process Execution Language (BPEL)**
  - XML-based Web Service composition language
  - modeling of Web Service based workflows
  - automatic deployment and execution on BPEL engine
Sample Process

- **Election Committee**
  - init_election
- **Registration Office**
  - set_candidates
  - set_number_of_voters
- **Elector**
  - vote
- **Attacker**
  - set_candidates

**BPEL Document**

**BPEL Engine**
Attack Effects

- sending one attack message
  → no effect
- sending 1000 attack messages (total traffic amount 0.5 MB)
  → full CPU load for more than 2 hours!
  → 350 MB additional memory usage
  → out-of-memory exceptions at BPEL engine

→ Denial-of-Service
→ *BPEL State Deviation* attack
Protection approach

- BPEL firewall
  - detecting non-protocol conforming messages
  - enforcing correct message sequence
- Sequence validation
  - BPEL document implies stateful communication protocol
  - firewall must know the current state inside the BPEL engine (comparable to stateful TCP firewall)

→ State automaton for message sequence validation
State automaton generation: concept

```xml
<sequence>
  <receive operation="init_election" />  
  <receive operation="set_candidates" />  
  <receive operation="set_number_of_voters" />  
  <while condition="vote_on()">
    <receive operation="vote" />  
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke operation="announce_winner" />  
    </case>
    <otherwise>
      <invoke operation="announce_no_winner" />  
    </otherwise>
  </switch>
</sequence>
```
Protection Concept

- `init_election`
- `set_candidates`
- `set_number_of_voters`
- `vote`
- `announce_winner`
- `announce_no_winner`
State automaton generation: example

<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
State automaton generation: example

<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
State automaton generation: example

```xml
<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
```
State automaton generation: example

<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
State automaton generation: example

```
<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
```
State automaton generation: example

<sequence>
  <receive op="init_election" />
  <receive op="set_candidates" />
  <receive op="set_number_of_voters" />
  <while condition="vote_on()">
    <receive op="vote" />
  </while>
  <switch>
    <case condition="not(equal(n1,n2))">
      <invoke op="announce_winner" />
    </case>
    <otherwise>
      <invoke op="announce_no_winner" />
    </otherwise>
  </switch>
</sequence>
State automaton generation: example

<sequence>
    <receive op="init_election" />
    <receive op="set_candidates" />
    <receive op="set_number_of_voters" />
    <while condition="vote_on()">
        <receive op="vote" />
    </while>
    <switch>
        <case condition="not(equal(n1,n2))">
            <invoke op="announce_winner" />
        </case>
        <otherwise>
            <invoke op="announce_no_winner" />
        </otherwise>
    </switch>
</sequence>
State automaton generation: example

<sequence>
  <receive op="init_election" />  
  <receive op="set_candidates" />  
  <receive op="set_number_of_voters" />  
  <while condition="vote_on()">  
    <receive op="vote" />  
  </while>  
  <switch>  
    <case condition="not(equal(n1,n2))">  
      <invoke op="announce_winner" />  
    </case>  
    <otherwise>  
      <invoke op="announce_no_winner" />  
    </otherwise>  
  </switch>  
</sequence>
Prototype Implementation

- State automaton generator
- Firewall using generated state automatons
- Evaluation:
  - No significant overhead for valid messages
  - Efficient detection and rejection of state deviation messages
    → no resource consumption at BPEL engine
Conclusion

- Workflow engines vulnerable to Denial of Service
- Firewall approach for fending attacks
- Protection effect bases on workflow-specific state automaton
- State automaton generated from BPEL document
- Prototype implementation illustrates protection performance
Thanks for your attention!