

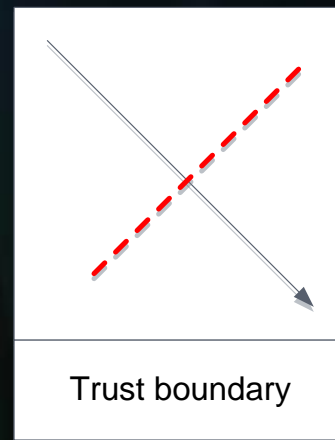
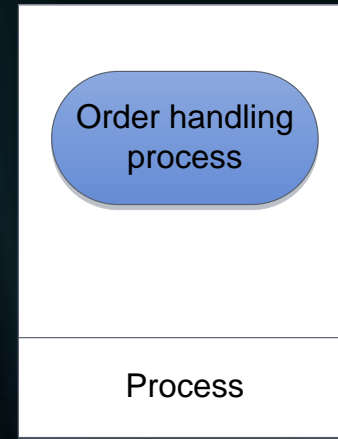
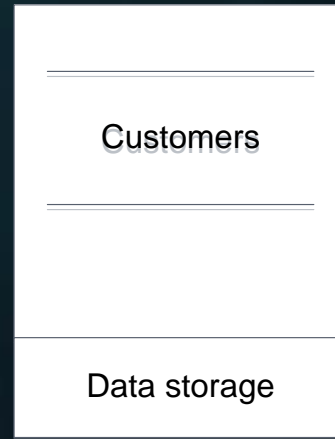
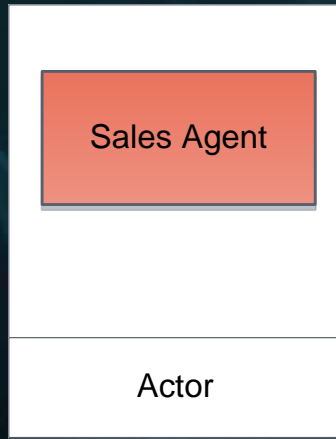
STRIDE Analyzer

Security Threat Analysis with PROLOG

Threat Modeling

- Used for distributed systems
- Based on data flow diagrams
- 5 different types of elements
- Different kind of threats per element

Elements



Simple Syntax Check

- Each node has to be from type Actor, Process or Storage

```
type(storage) .
```

```
type(process) .
```

```
type(actor) .
```

- Arcs connect nodes that really exists

Invalid node:

```
node(custTable, csvfile) .
```

Invalid arc

```
node(logger, process) .
```

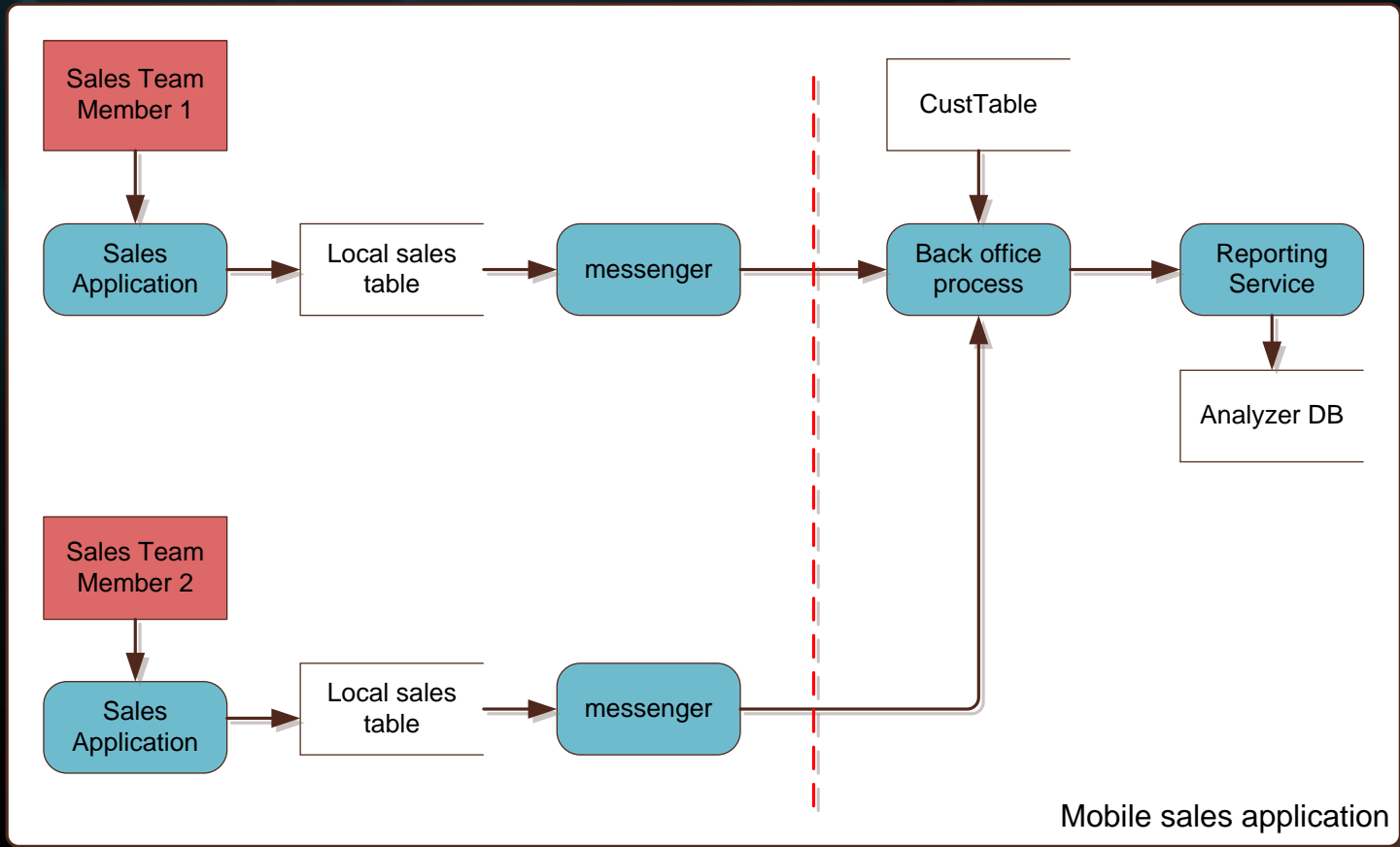
```
node(logfile, storage) .
```

```
arc(logger, printer) .
```

Simple Syntax Check

```
/* node has wrong type */  
wrongType(X):-node(X,T),not(type(T)).  
/* invalid connection */  
wrongArc(S,T):-arc(S,_),not(node(S,_)).  
wrongArc(S,T):-arc(_,T),not(node(T,_)).  
  
syntaxError:-wrongType(_),!.  
syntaxError:-wrongArc(_,_),!.
```

First Try



First Try

Nodes:

```
node (sales001,actor) .
node (sales002,actor) .
node (salesApp1,process) .
node (salesApp2,process) .
node (localSalesTable1,storage) .
node (localSalesTable2,storage) .
node (messenger1,process) .
node (messenger2,process) .
node (salesAppServer,process) .
node (custTable,storage) .
node (reportingService,process) .
node (statistik,storage) .
```

Data flow:

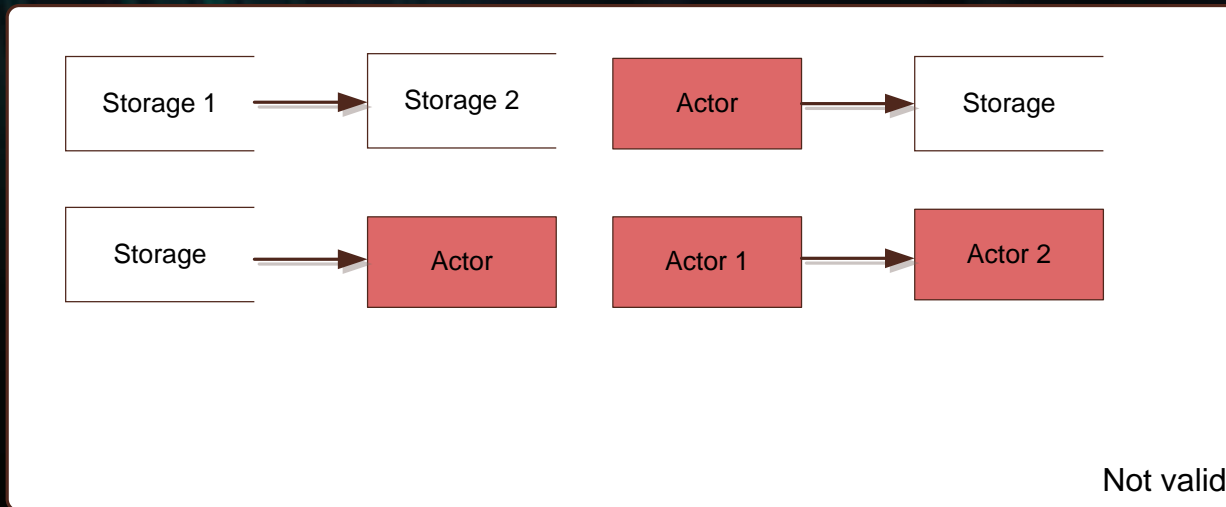
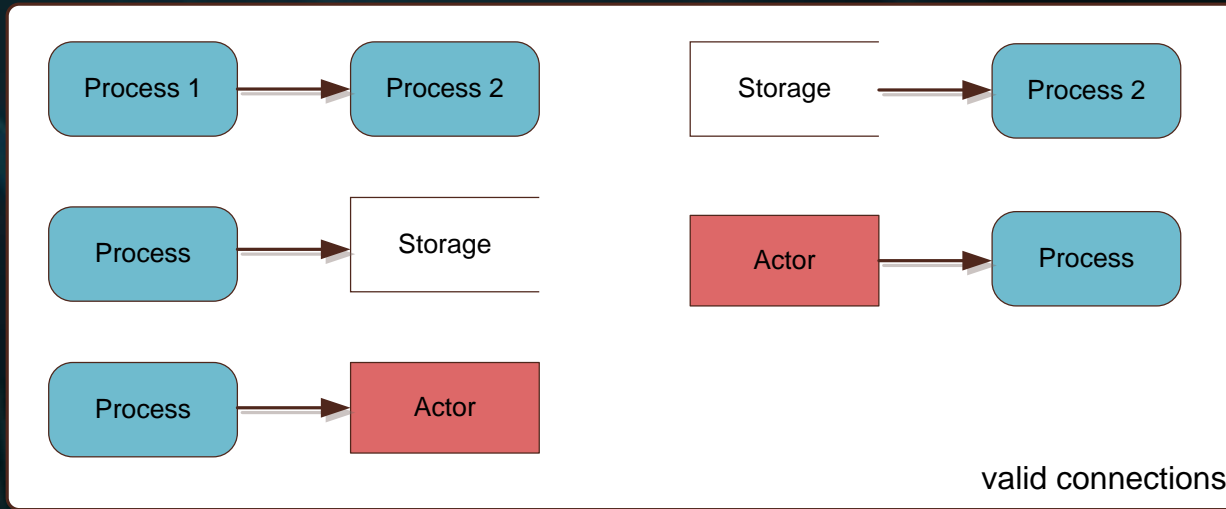
```
arc (sales001,salesApp1) .
arc (sales002,salesApp2) .
arc (salesApp1,localSalesTable1) .
arc (salesApp2,localSalesTable2) .
arc (localSalesTable1,messenger1) .
arc (localSalesTable2,messenger2) .
arc (messenger1,salesAppServer) .
arc (messenger2,salesAppServer) .
arc (custTable,salesAppServer) .
arc (salesAppServer,reportingService) .
arc (reportingService,statistik) .
```

230 ?- syntaxError.

No



Additional Constrains (1)



Check with Additional Constrains

```
wrongType (X) :-node (X,T) , not (type (T)) .
wrongArc (S,T) :-arc (S,T) , not (node (S,_)) , node (T,_) .
wrongArc (S,T) :-arc (S,T) , node (S,_) , not (node (T,_)) .
wrongArc (S,T) :-arc (S,T) , not (node (S,_)) , not (node (T,_)) .

/* Additional Constraints */
wrongArc (S,T) :-arc (S,T) , node (S,storage) , node (T,storage) .
wrongArc (S,T) :-arc (S,T) , node (S,storage) , node (T,actor) .
wrongArc (S,T) :-arc (S,T) , node (S,actor) , node (T,storage) .
wrongArc (S,T) :-arc (S,T) , node (S,actor) , node (T,actor) .

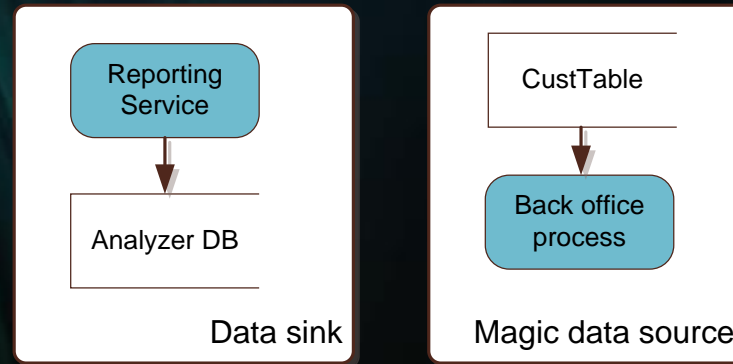
syntaxError :-wrongType (_), ! .
syntaxError :-wrongArc (_,_) , ! .
```

Additional Constraints (2)

Be aware of magic data sources and data sinks. Data doesn't exist per se, it has to be create somewhere. If you ignore this fact, you ignore a potential risk.

Example:

- Customer data holds information like credit limit and it is used inside your application.
- Is the analyzer DB really never used ?



Additional Constraints (2)

```
/* no writing process */
```

```
magicDB(DB) :- node(DB, storage), not(arc(_, DB)).
```

```
magicDB(DB) :- node(DB, storage),  
               arc(WRITER, DB),  
               not(node(WRITER, process)).
```

```
/* data is never read */
```

```
dataSink(DB) :- node(DB, storage), not(arc(DB, _)).
```

```
/* SYNTAX CHECK */
```

```
wrongType(X) :-node(X,T),not(type(T)).
```

```
wrongArc(S,T) :-arc(S,T),not(node(S,_)),node(T,_).
```

```
wrongArc(S,T) :-arc(S,T),node(S,_),not(node(T,_)).
```

```
wrongArc(S,T) :-arc(S,T),not(node(S,_)),not(node(T,_)).
```

```
%% Additional constraints
```

```
wrongArc(S,T) :-arc(S,T),node(S,storage),node(T,storage).
```

```
wrongArc(S,T) :-arc(S,T),node(S,storage),node(T,actor).
```

```
wrongArc(S,T) :-arc(S,T),node(S,actor),node(T,storage).
```

```
wrongArc(S,T) :-arc(S,T),node(S,storage),node(T,storage).
```

```
wrongArc(S,T) :-arc(S,T),node(S,actor),node(T,actor).
```

```
%% data base constraints
```

```
magicDB(DB) :-node(DB,storage),not(arc(_,DB)).
```

```
magicDB(DB) :-node(DB,storage),arc(WRITER,DB),  
not(node(WRITER,process)).
```

```
dataSink(DB) :-node(DB,storage),not(arc(DB,_)).
```

```
syntaxError:-wrongType(_),!.
```

```
syntaxError:-wrongArc(_,_),!.
```

```
syntaxError:-magicDB(_),!.
```

```
syntaxError:-dataSink(_),!.
```

Check first example

39 ?- wrongType(X).

No

40 ?- wrongArc(S,T).

No

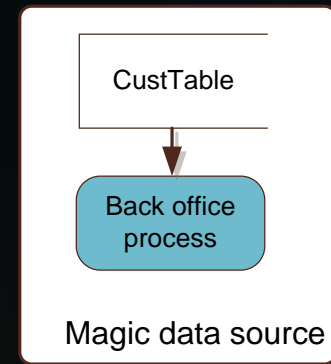
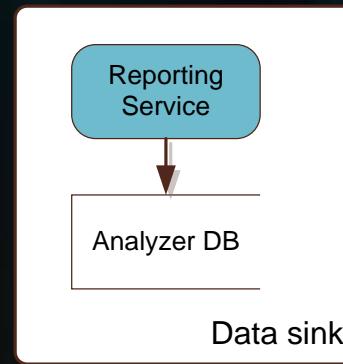
41 ?- magicDB(X).

X = **custTable** ;

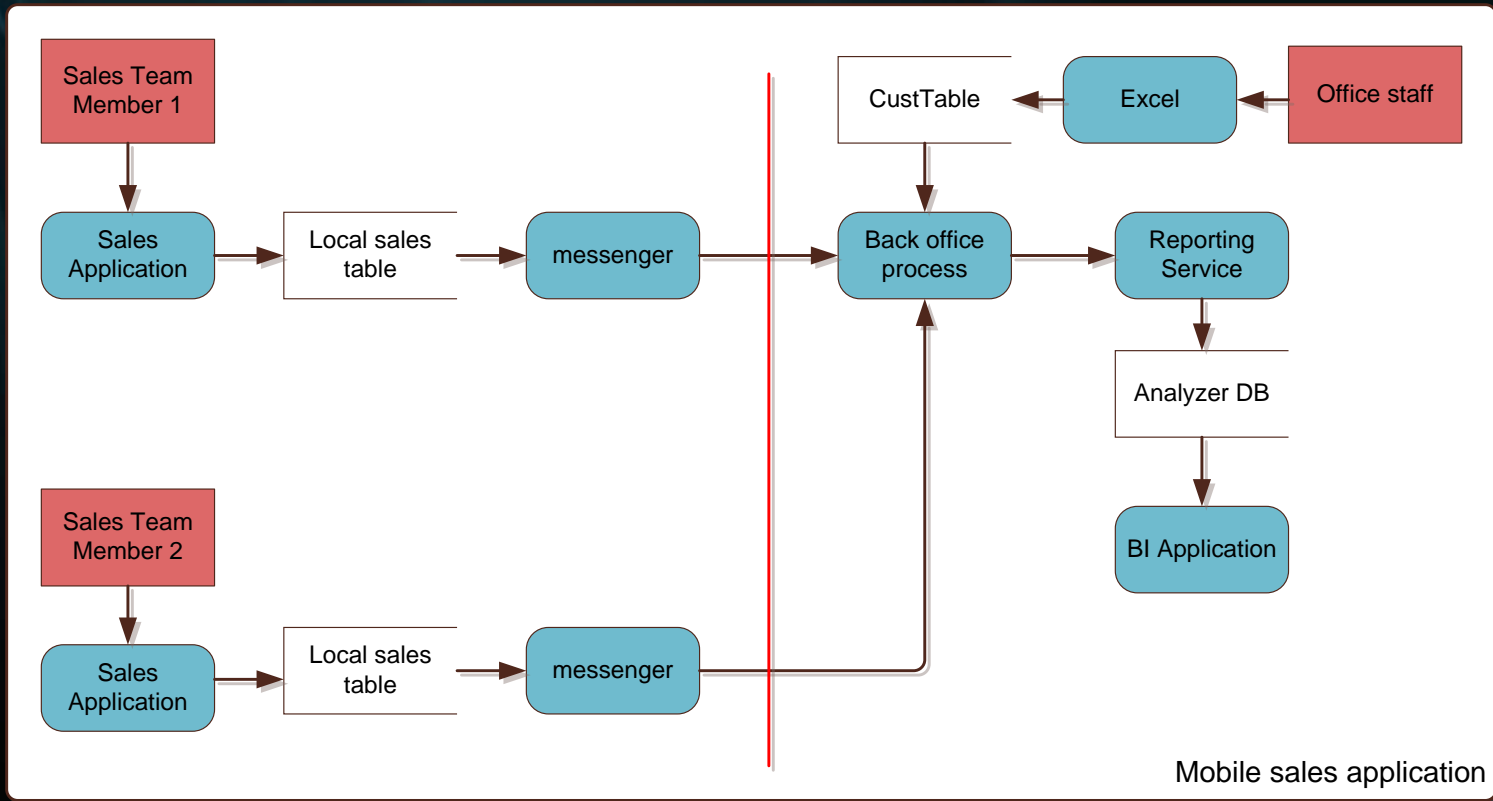
No

42 ?- dataSink(X).

X = **statistik**



First Example repaired

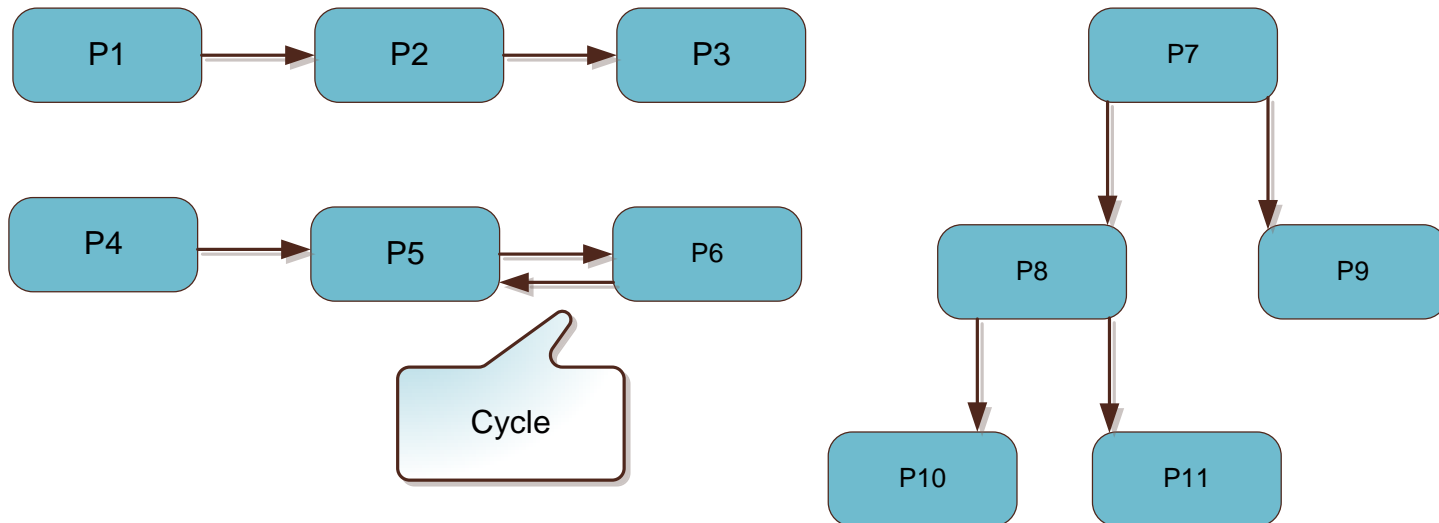


```
node(officestaff,actor).  
node(excel,process).  
node(biApplication,process).
```

```
arc(officestaff,excel).  
arc(excel,custTable).  
arc(statistik,biApplication).
```

Reachability

- For one special node, say who writes it ?
Say for one interesting node, like a sales process engine, a web server, etc. who sends data to this node. Also consider transitive data flow.
- For one special node, say who reads its data ?
If this node is controlled by an attacker, who reads data from it / to whom is data sent ?



Example

Reachability

```
/* data flow from X to .. */
writeNodes(X):-go(X, []).
go(X,T) :- not(arc(X,_)),fail.
go(X,T) :- arc(X,N),not(member(N,T)),display(N),nl,go(N, [X|T]).

/* data flow from .. to X */
readNodes(X):-go2(X, []).
go2(X,T) :- not(arc(_,X)),fail.
go2(X,T) :- arc(P,X),not(member(P,T)),display(P),nl,go2(P, [X|T]).
```

```
1 ?- writeNodes(p1).           4 ?- readNodes(p3).
p2                             p2
p3                             p1
```

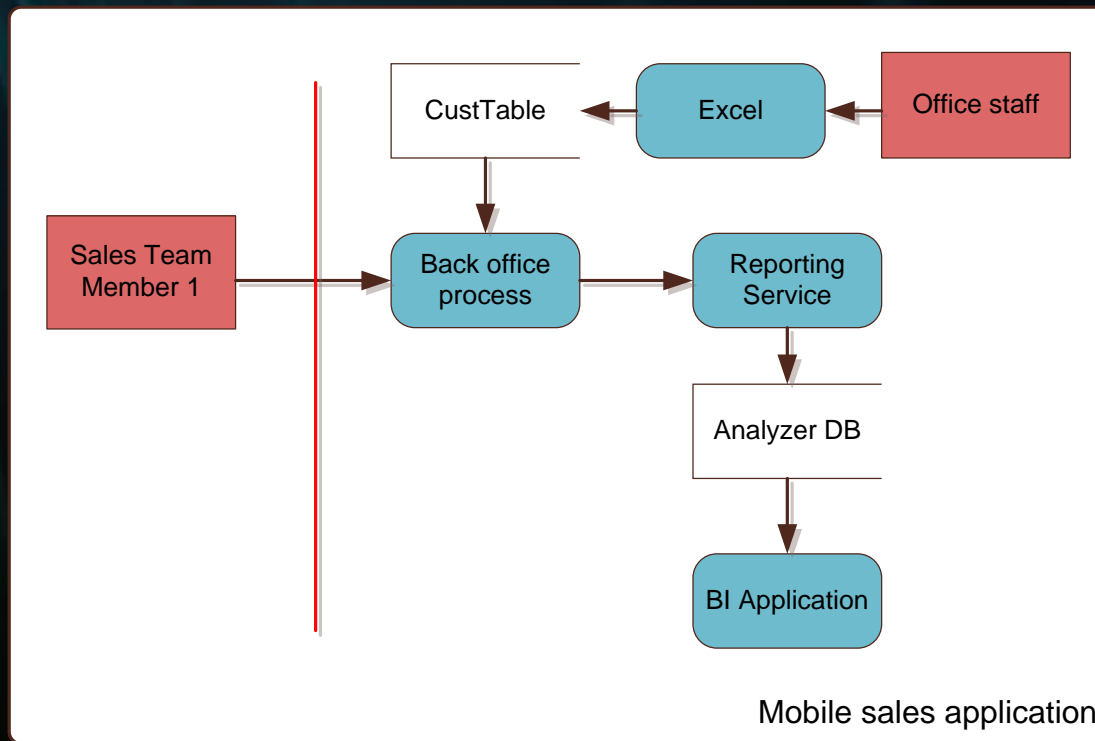
```
2 ?- writeNodes(p4).           5 ?- readNodes(p5).
p5                             p4
p6                             p6
```

```
3 ?- writeNodes(p7).           6 ?- readNodes(p11).
p8                             p8
p9                             p7
p10
p11
```



Second Try

Focus on one part of your application inside your trust boundaries. If you model a server side application, you cannot say why and where from some data is sent to you. An attacker will surely not use your client application.



Trust boundary

New predicate `boundary(SOURCE, TARGET)` where you don't trust transmitted information. Like data sent over the internet or non encrypted WLAN.

```
node(sales001,actor).
node(backoffice,process).
node(custTable,storage).
node(excel,process).
node(office001,actor).
node(reportingService,process).
node(statistik,storage).
node(biApplication,process).
arc(sales001,backoffice).
arc(office001,excel).
arc(excel,custTable).
arc(custTable,backoffice).
arc(backoffice,reportingService).
arc(reportingService,statistik).
arc(statistik,biApplication).
boundary(sales001,backOffice).
```

```
%% wrong boundary
wrongBoundary(X,_):-not(node(X,_)). % source not a node
wrongBoundary(_,X):-not(node(X,_)). % target not a node
```

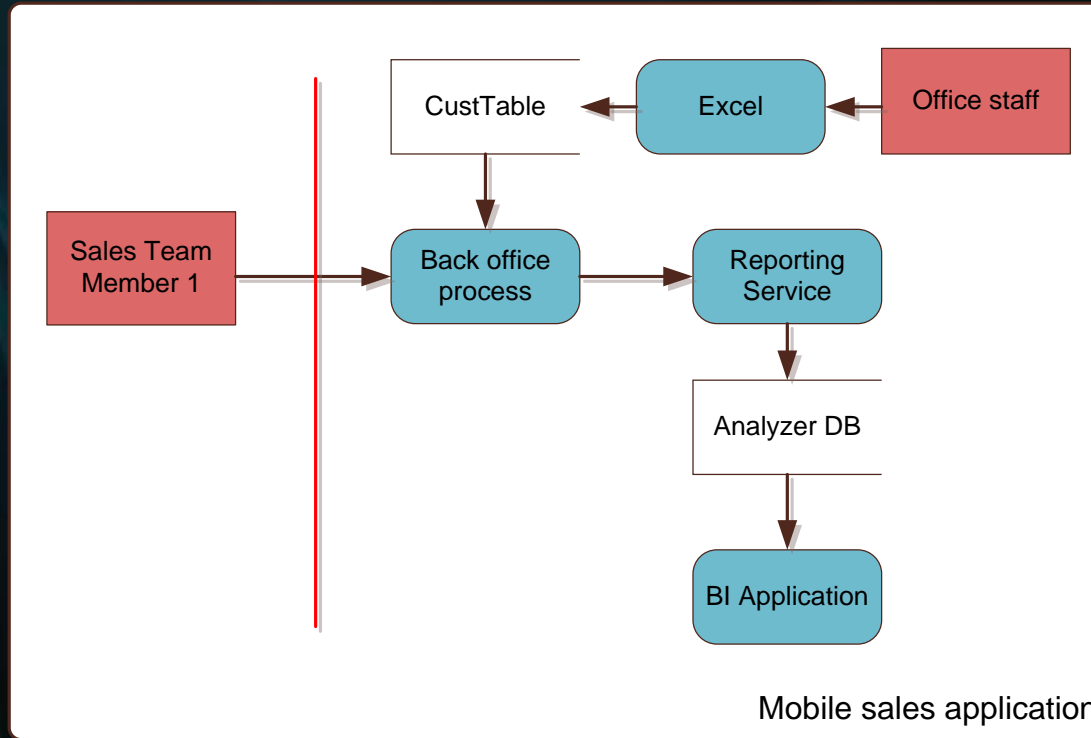
```
syntaxError:-wrongBoundary(_,_).
```

Trust boundary

```
readUntrustedNodes(X):-go3(X,[],no).

go3(X,_,_) :- not(arc(_,X)),fail.
/* trust boundary has already been crossed */
go3(X,T,yes):- arc(P,X),
                not(member(P,T)),
                display(P),nl,
                go3(P,[X|T],yes).
/* trust boundary has not been crossed
   and previous node is not outside */
go3(X,T,no):- arc(P,X),
               not(boundary(P,X)),
               not(member(P,T)),
               go3(P,[X|T],no).
/* trust boundary will be crossed */
go3(X,T,no):- arc(P,X),
               boundary(P,X),
               not(member(P,T)),
               display(P),nl,
               go3(P,[X|T],yes).
```

Trust Boundary



```
1 ?- readUntrustedNodes(backoffice) .  
sales001
```

```
2 ?- readUntrustedNodes(biApplication) .  
sales001
```



Further Readings

- Aufdecken von Fehlern im Sicherheitsentwurf mithilfe des STRIDE-Ansatzes
<http://msdn.microsoft.com/msdnmag/issues/06/11/ThreatModeling/default.aspx?loc=de>
- Threat Modeling, Frank Swiderski & Window Snyder, Microsoft Press, 2004
http://www.amazon.de/Threat-Modeling-Microsoft-Professional-Swiderski/dp/0735619913/ref=sr_1_1?ie=UTF8&s=books-intl-de&qid=1199962814&sr=1-1
- <http://www.erpcoder.at/stride>