Mohamed Almorsy, John Grundy, and Amani S. Ibrahim

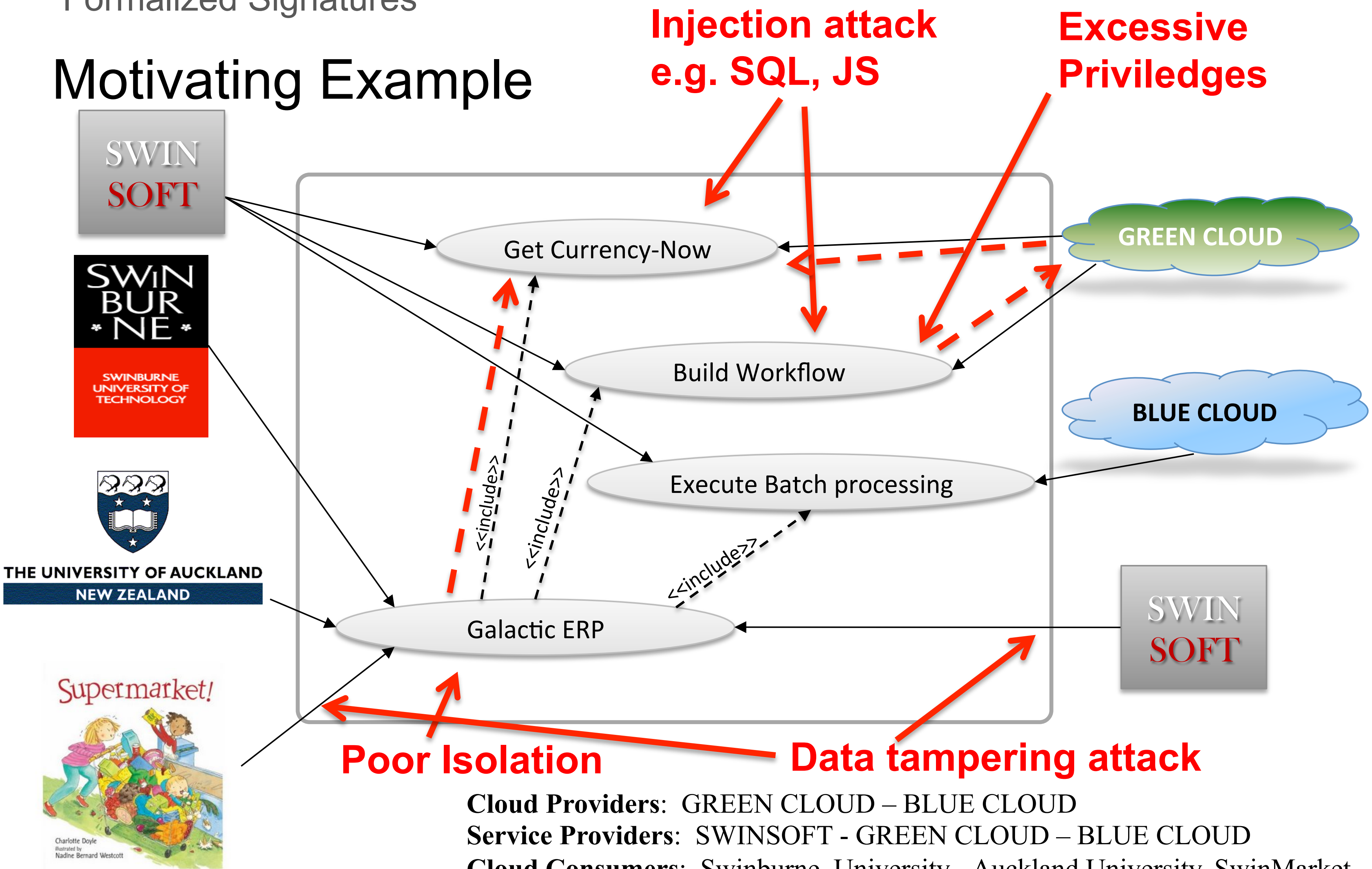
Automated Software Architecture Security Risk Analysis Using Formalized Signatures

SWINBURNE UNIVERSITY OF TECHNOLOGY

Outline

- Motivation
- Weaknesses and Security Metrics
- Approach
- Example
- Evaluation
- Conclusions

Motivating Example



Cloud Providers: GREEN CLOUD – BLUE CLOUD
Service Providers: SWINSOFT - GREEN CLOUD – BLUE CLOUD
Cloud Consumers: Swinburne University- Auckland University, SwinMarket

Architecture Security Analysis & Attacks

- Architecture Security Analysis e.g.
  - MS STRIDE, EOP Card Game, CAPEC
- Common attacks to look for
  - Man in the middle
  - Denial of service
  - Data tampering
  - Injection attack
  - …
- BUT – what indicates a system might be vulnerable to such attacks?? E.g. consider the previous example!

System Architecture level security metrics

- Attack surface – proportion of system attacker could use/ access to attack system
- Compartmentalization – isolation of system components from each other to minimise cross-compromise
- Least privilege – should grant components minimal privileges to carry out operation
- Secure failure – if something goes wrong, don't expose sensitive details of system
- Security isolation – between components

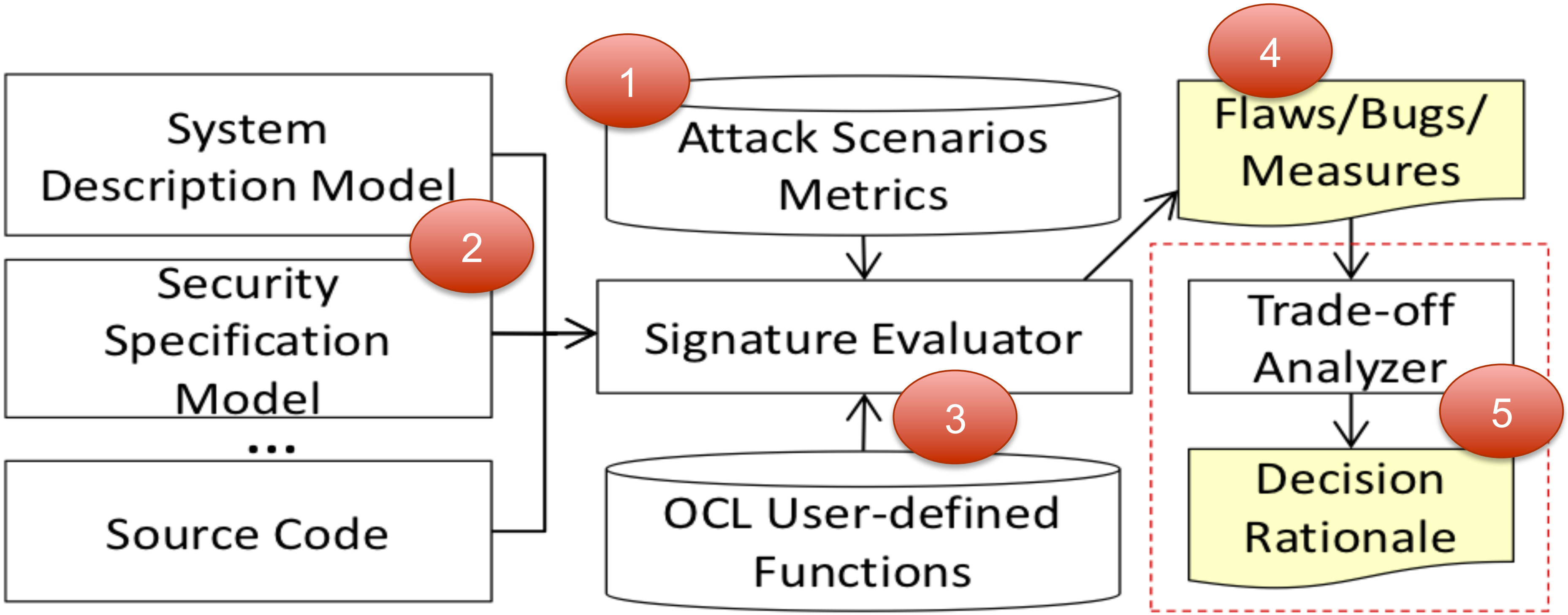
Key research questions

- Can we use security metrics to identify weaknesses?
- Can we identify these weaknesses from architectural-level characteristics and structures of a cloud application?
- Can we formalise currently informal weakness and metric definitions e.g. CAPEC database to make them amenable for automated architectural analysis?
- Can we use the identified weaknesses / vulnerabilities to alert cloud/service providers and/or cloud consumers to actual or possible security problems?
- Can we use this information to mitigate the problems?

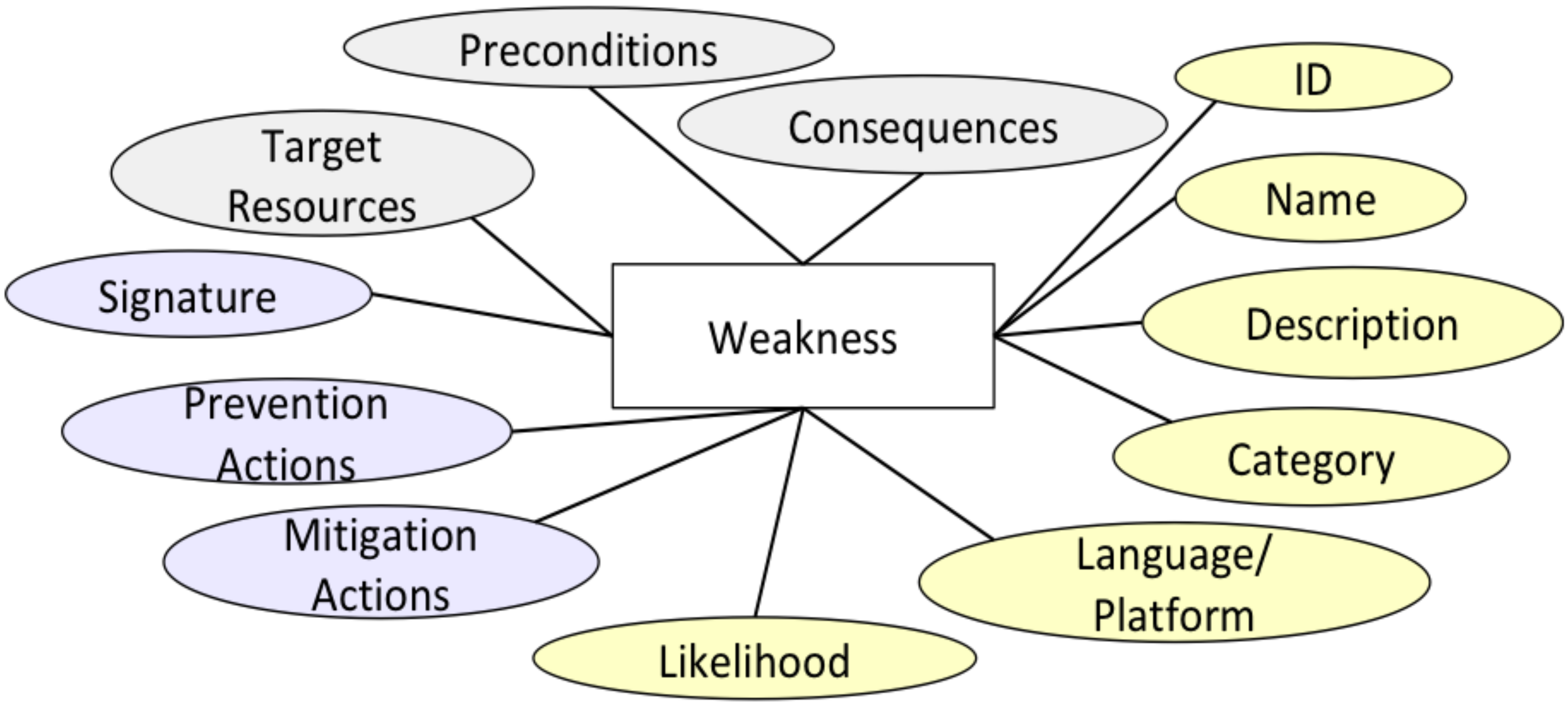
Our Approach

- Previously, we described code-level “vulnerability signatures” used to detect via static analysis (ASE 2012)
- Now we have looked for signatures & metrics to indicate weaknesses @ architecture levels
- Formalise CAPEC attack pattern signatures, architecture vulnerabilities, security metrics via OCL
- Search for matching signatures in system architecture & security requirements definitions
- Perform trade-off analysis of vulnerabilities/mitigations
- Apply mitigations to address weaknesses

Process


System Description Model
Security Specification Model
...
Source Code
Attack Scenarios Metrics
Signature Evaluator
OCL User-defined Functions
Flaws/Bugs/ Measures
Trade-off Analyzer
Decision Rationale
1
2
3
4
5

1. “Weakness Definitions”



Examples (Simplified)

Any two components that communicate through an unencrypted channel and one or both operate in untrusted zone or do not apply cryptography controls on communicated messages

Any publicly accessible component that does not operate input sanitization control (or an application firewall), and does not have authentication control

Any component that is deployed on an untrusted host (malicious insider) or zone, sends data in plain text, or does not operate authorization control.

ID

1

```
context System inv Man-in-the-Middle Attack:
self.components->select(C1|
        C1.DeploymentZoneType = 'Untrusted'
  and self.components.exists(C2 |
        C2.Channels->exists(Ch |
           Ch.TargetComponent = C1
      and Ch.EncryptionControlDeployed = fa
  and C1.EncryptionControlDeployed = false
  and C2.EncryptionControlDeployed = false))
```

Any two components that communicate through an unencrypted chan
their communicated messages.

2

```
context System inv Denial-of-Service Attack:
self.components->select(C1|
      C1.DeploymentZoneType = 'Untrusted'
 and  C1.AuthenticationControlDeployed = fal
 and  (C1.InputSanitizationControlDeployed =
   or C1.Host.Network.FirewallControlDeploye
```

Any publicly accessible component that does not operate input saniti

3

```
context System inv DataTampering:
self.components->select(C1|
       C1.DeploymentZoneType = 'Untrusted'
 and self.components.exists(C2 |
          C2.Channels->exists(Ch |
              Ch.TargetComponent = C1
          and Ch.EncryptionControlDeployed = false)
      and C1.EncryptionControlDeployed = false
      and C2.EncryptionControlDeployed = false))
```

Any component that is deployed on an untrusted host (malicious insider) or zone, sends data in plain text, or does not operate authorization control.

Examples (2) – some metrics

4	context System inv **AttackSurface**: self.components->select(C1| C1. DeploymentZoneType = 'Untrusted')->collect(C2 | C2.Functions)->size()
Number of the functions defined in the provided interfaces of the public system components and number of functions defined in the required interfaces of the system public components that are used by other components.	
5	context System inv **Compartmentalization**: self.components->select(C | C.AuthenticationControlDeployed = tru and C.AuthorizationControlDeployed = true
Number of architecture components that apply Authn. and Authz. c	
6	context System inv **FailSecurely**: self.components->collect(C | C.Functions->s F.IsCritical = true)->siz self.components->collect(C |C.Functions- F.IsCritical = true)->size())
The average of critical methods and attributes in each system co	
7	context System inv **Defense-in-depth**: self.select(C | C.IsCritical= tru and C.AuthenticationControlDeployed = true and C.AuthorizationControlDeployed = true and C.CryptographyControlDeployed = true and C.Host.AuthenticationControlDeployed = true and C. Host.AuthorizationControlDeployed = true and C. Host.CryptographyControl = true)->size() / self.select(C | C.IsCritical = true)->size()
The ratio of critical components that have layered security compared to the total number of critical components in the system.	

# of functions defined in interfaces of public system components that are used by other components

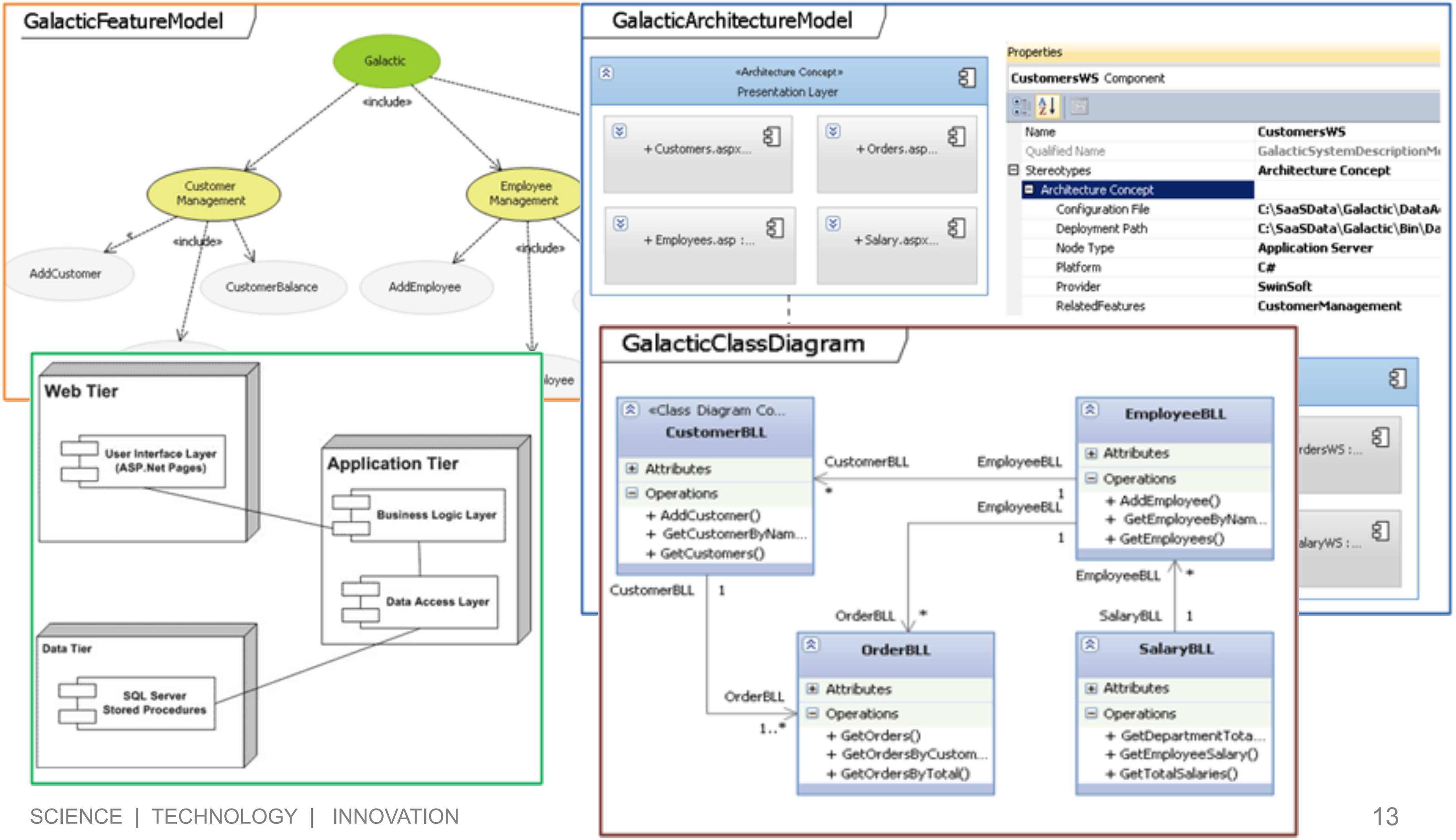
# of components that apply Authentication

The ratio of critical components that have layered security compared to the total number of critical components in the system

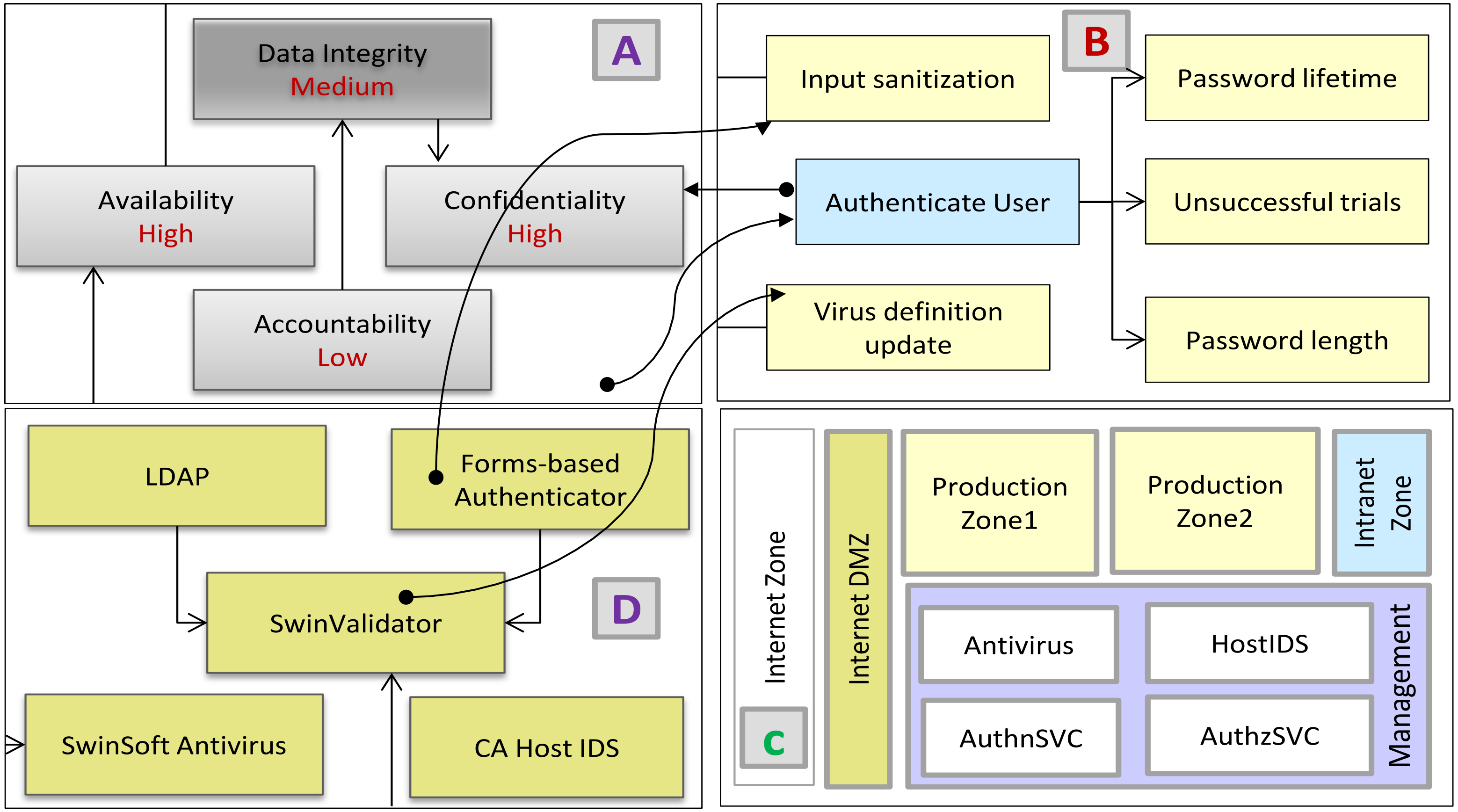
2. Models to check & the Analysis process

- System Description Model
- Security Specification Model
- System-Security Mappings
- Signature Evaluator – vulnerabilities & metrics
- Results
- Trade-off analysis

Some source models for architecture

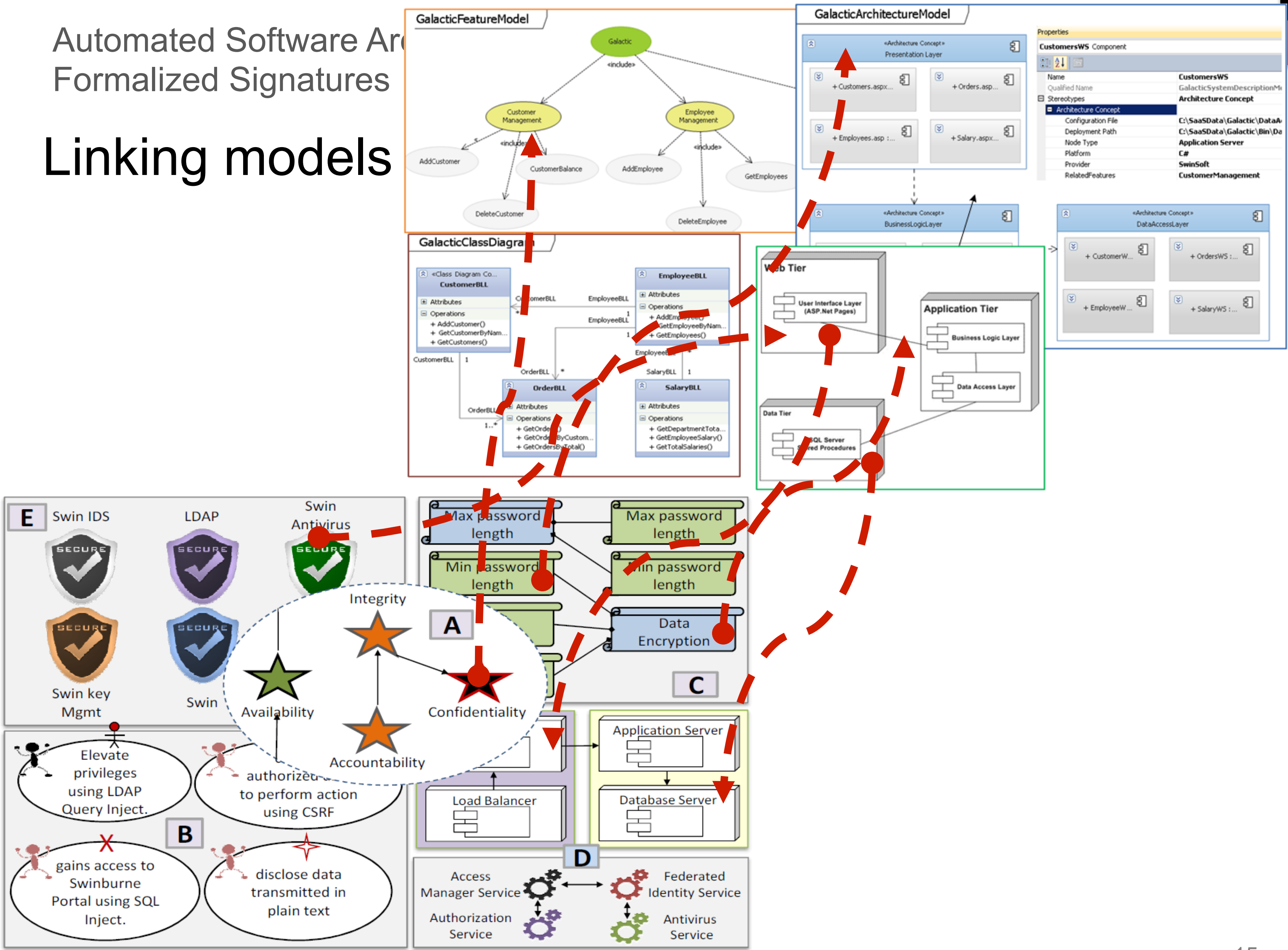


Some security specification models



Automated Software Ar
Formalized Signatures

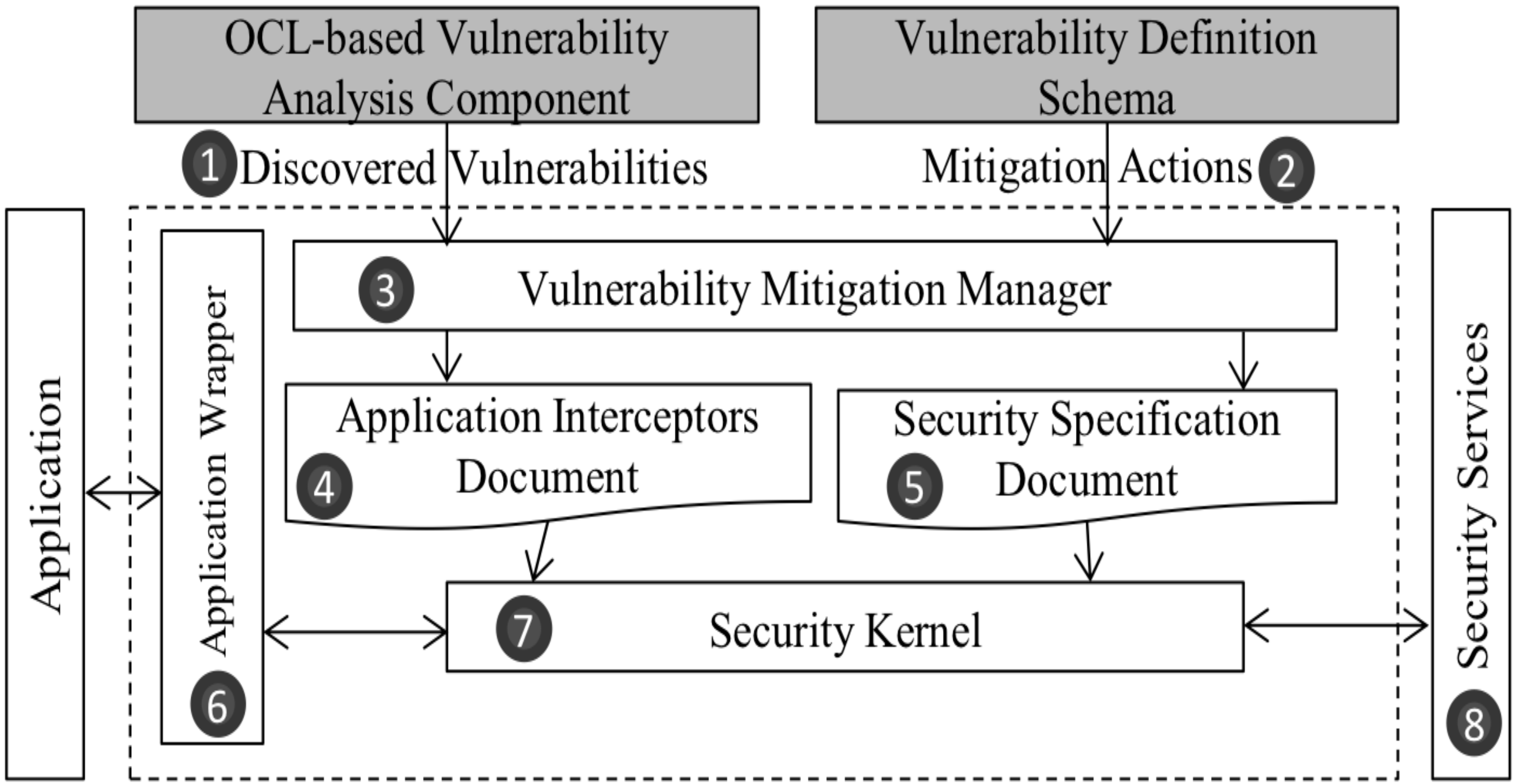
Linking models



3. Architecture Analysis & Weakness Mitigation

- Mappings between architecture (system) <-> security specification => desired security levels, constraints
- Signatures encoded in OCL => vulnerable components/ connectors to search for & metrics to take
- OCL Signatures compiled => set of C# functions
- C# functions run over system/security model => locate potential weaknesses/vulnerabilities in architecture
- Trade-off analysis using mitigation information => possible changes to system and/or security models…
- Run-time application of vulnerability mitigations…

Using the vulnerabilities / mitigations @ runtime…



Evaluation

Benchmark	Downloads	KLOC	Files	Comps	Classes	Method
BlogEngine	>46,000	25.7	151	2	258	616
BugTracer	>500	10	19	2	298	223
Galactic	-	16.2	99	6	101	473
KOOBOO	>2,000	112	1178	13	7851	5083
NopCommerce	>10 Rel.	442	3781	8	5127	9110
SplendidCRM	>400	245	816	7	6177	6107

- 5 open source + our own motivating scenario
- Various levels of complexity, architecture, implemented security models
- Four attack scenarios: Man-in-The-Middle, Denial of Service, Data Tampering, and Injection attacks

Swinburne

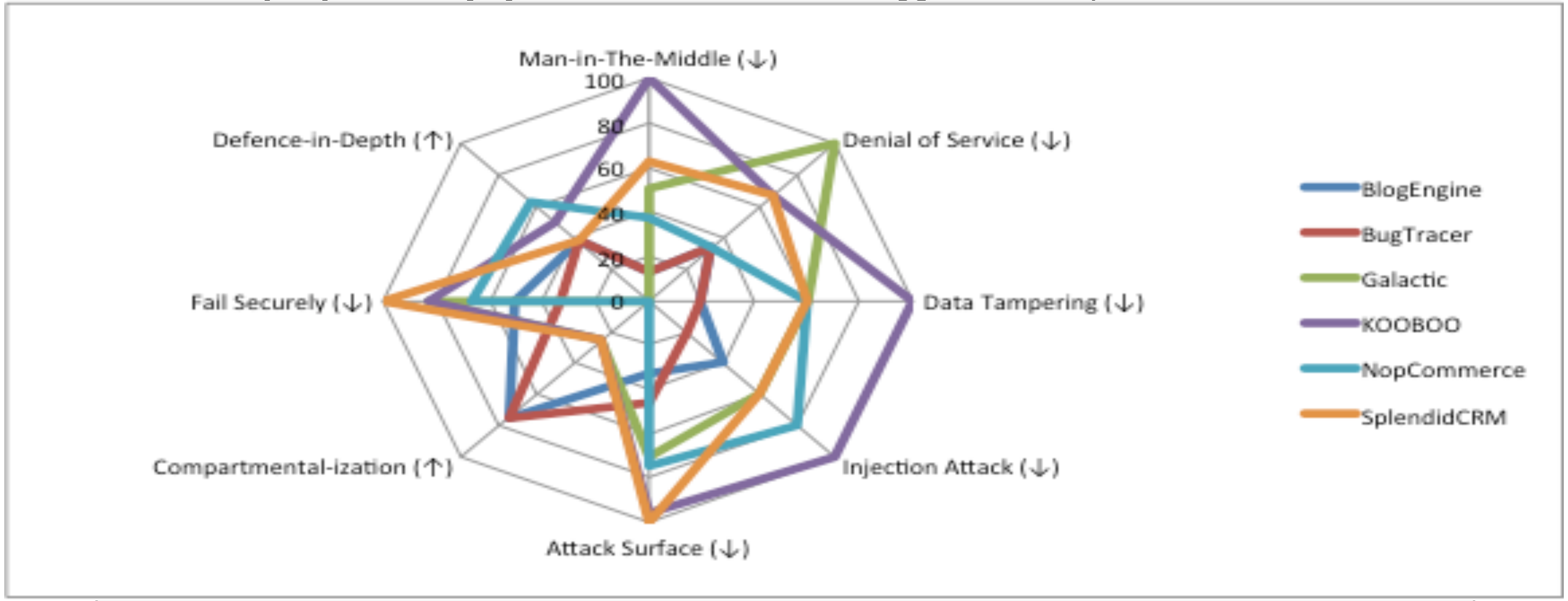
Results

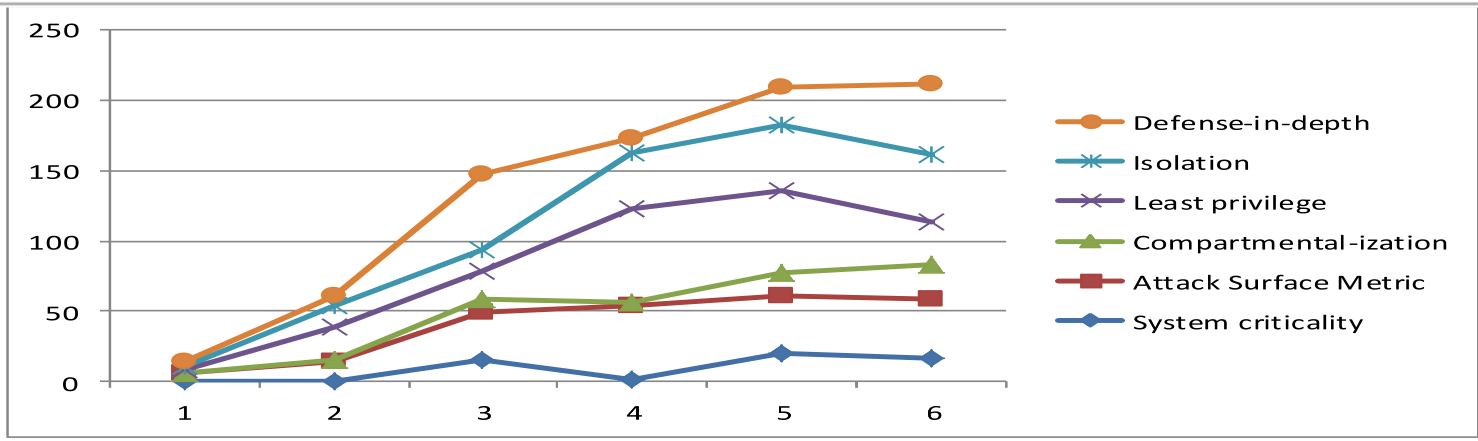
Scenario / Metric D = DISCOVERED FLAWS FP= FALSE POSITIVES FN = FALSE NEGATIVES ↓ => lower = better		BlogEngine	BugTracker	Galactic	KOOBOO	NopCommerce	SplendidCRM	Total
Security Scenarios								
Man-in-The-Middle (↓)	**D**	1	1	4	8	3	5	22
	FP	0	0	0	1	0	0	1
	FN	0	0	0	1	0	1	2
Denial of Service (↓)	**D**	1	1	3	2	1	2	10
	FP	0	0	0	0	0	1	1
	FN	0	0	0	1	1	0	2
Data Tampering (↓)	**D**	1	1	3	5	3	3	16
	FP	0	0	0	2	0	0	2
	FN	0	0	1	0	1	0	2
Injection Attack (↓)	**D**	2	1	3	5	4	3	18
	FP	0	0	1	1	0	1	3
	FN	0	1	1	1	0	0	3
Total	**D**	5	4	13	20	11	13	66
	FP	0	0	1	4	0	2	7
	FN	0	1	2	3	2	1	9
Average Precision = 90% Average Recall = 87% F-Measure = 88%								

Results (2)

Scenario / Metric M = METRIC MEASURED VALU,E FP= ALSE POSITIVES FN = FALSE NEGATIVES ↑ => higher is better; ↓ => lower is better		BlogEngine	BugTracker	Galactic	KOOBOO	NopCommerce	SplendidCRM	Total
Security Metrics								
Attack Surface (↓)	**M**	8	11	17	23	18	24	101
	FP	1	2	2	1	2	4	12
	FN	0	0	1	3	2	1	7
Compartmental-ization (↑)	**M**	1	1	3	3	4	3	14
	FP	0	0	0	0	1	0	1
	FN	0	0	1	1	0	0	2
Fail Securely (↓)	**M**	0.3	0.2	0.5	0.5	0.4	0.6	-
	FP	2	1	0	0	0	1	4
	FN	1	0	0	0	1	1	3
Defence-in-Depth (↑)	**M**	0.5	0.5	0.8	0.4	0.3	0.5	-
	FP	0	1	0	0	1	0	2
	FN	0	2	0	1	0	1	4
Average Precision = 91% Average Recall = 89% F-Measure = 90%								

Results (3) – Apples vs Oranges ☺; Performance





All is not what it may seem - some things to note…

- Can compare systems in the same domain – but appearances can be (very) deceiving…
- Vulnerability Counts vs Metrics vs meaning
  - need to compare like with like
  - Criticality of the issue vs simple occurrences
  - System scale makes a large difference
- Just one critical weakness can cause whole system to be compromised under attack; lots of minor weaknesses may be tolerable
- Its rather slow to analyse many of these => non-real time
- Change to environment / co-deployed services/applications => changes to measures / counts…

Conclusions, Future work

- A range of architecture vulnerabilities and security metrics can be formalised
- These formalised specifications can be used to check architecture security properties and vulnerabilities
- Applying to range of open source applications shows the technique finds a number of vulnerabilities present in the applications
- Authoring these specifications is hard
- Technique relies heavily on soundness of specifications
- Some vulnerabilities need dynamic analysis to find
- Interpretation of measures / counts; criticality of flaws

Automated Software Architecture Security Risk Analysis Using Formalized Signatures

Thanks!

Questions?

References

Almorsy, M., Ibrahim, A., Grundy, J.C., Adaptive Security Management in SaaS Applications, Chapter 8 in Security, Privacy and Trust in Cloud Systems, Springer, 2013.

Almorsy, M., Grundy, J.C. and Ibrahim, A., Automated Software Architecture Security Risk Analysis Using Formalized Signatures, 2013 IEEE/ACM International Conference on Software Engineering (ICSE 2013), San Franciso, May 2013, IEEE CS Press

Almorsy, M., Grundy, J.C. and Ibrahim, A. Supporting Automated Vulnerability Analysis using Formalized Vulnerability Signatures, 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012), Sept 3-7 2012, Essen, Germany, ACM Press.

Almorsy, M., Grundy, J.C. and Ibrahim, A., Supporting Automated Software Re-Engineering Using "Re-Aspects", 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012), Sept 3-7 2012, Essen, Germany, ACM Press.

Almorsy, M., Grundy, J.C. and Ibrahim, I., VAM-aaS: Online Cloud Services Security Vulnerability Analysis and Mitigation-as-a-Service, 2012 International Conference on Web Information Systems Engineering (WISE 2012), Nov 28-30 2012, Paphos, Cyprus, LNCS, Springer.

Almorsy, M., Grundy, J.C. and Ibrahim, A., MDSE@R: Model-Driven Security Engineering at Runtime, 4th International Symposium on Cyberspace Safety and Security (CSS 2012), Melbourne, Australia, Dec 12-13 2012, Springer.

Almorsy, M., Grundy, J.C., Ibrahim, A., SMURF: Supporting Multi-tenancy Using Re-Aspects Framework, 17th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2012), Paris, France, July 2012, IEEE CS Press.

Almorsy, M. and Grundy, J.C. TOSSMA: A Tenant-Oriented SaaS Security Management Architecture, 5th IEEE Conference on Cloud Computing (CLOUD 2012), IEEE CS Press, Waikiki, Hawai, USA, June 24-29 2012.

SWINBURNE UNIVERSITY OF TECHNOLOGY