

Flanking Security in Critical Infrastructures

George Stergiopoulos

ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ



ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

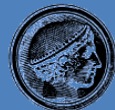
Flanking security in Critical Infrastructures:

Time-based impact analysis

Risk mitigation

Detection of logical error in functionality

George Stergiopoulos



Department of Informatics
Athens University of Economics and Business

OUTLINE

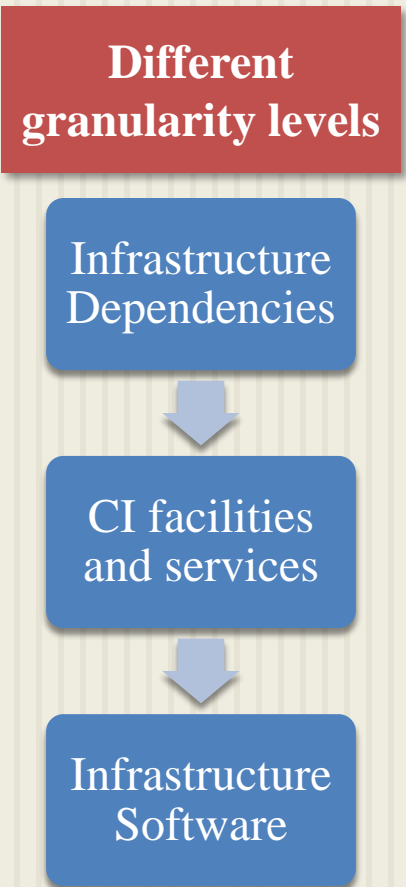
3

- Background
 - ▣ Basic Concepts
 - ▣ Research motivation
 - ▣ Existing approaches
 - ▣ Thesis' Objectives
- Contribution
 - ▣ Time-based analysis of impact
 - ▣ Risk mitigation using graph theory
 - ▣ Automatic detection of logical errors
- Conclusion
 - ▣ Publications
 - ▣ Future work

BASIC CONCEPTS

4

- ❑ **Critical Infrastructures (CIs):** “*Physical and information technology facilities, networks, services and assets which, if disrupted or destroyed, would have a serious impact on the health, safety, security or economic well-being*”.
- ❑ **Interdependency:** The state of one CI depends on the output (product, asset or service) of another. [Rinaldi, 2001]
- ❑ **Software logical errors:** Software flaws due to erroneous implementation of functionality into source code. [Felmtseger, 2010]



RESEARCH MOTIVATION

5

□ Overlooked characteristics in CIs:

1. Static analysis of impact between infrastructures.
 - Need to analyze evolution of impact to propose actions
2. Absence of risk mitigation mechanisms.
 - Need to pinpoint points-of-failure in dependency graphs
3. Human error critical in high-impact failures.
 - Use of third party software in CIs.



EXISTING APPROACHES

6/25

CI interdependency analysis

- ❑ Sector-specific tools.
 - ▣ Mostly energy and water.
- ❑ Analysis best on economic relationships.
 - ▣ Purely economical.

Software functionality

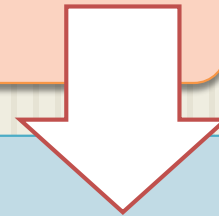
- ❑ Detect a priori flaws.
 - ▣ e.g. null pointers, buffer overflows.
- ❑ Debugging techniques.
 - ▣ No automated program analysis.
 - ▣ Only detect specific types of flaws.

THESIS' OBJECTIVES

7

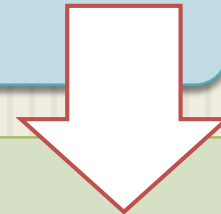
1. Time-based impact evolution in dependencies

- Growth rates of failures



2. Risk mitigation using graph theory

- *Pinpoint CIs and mitigate risk*



3. Logical errors in software

- Functionality flaws lead to failures in infrastructures

TIME-BASED IMPACT EVOLUTION IN DEPENDENCIES

8

1. Time-based impact evolution in dependencies

- Growth rates of failures

2. Risk mitigation using graph theory

- *Pinpoint CIs and mitigate risk*

3. Logical errors in software

- Functionality flaws lead to failures in infrastructures

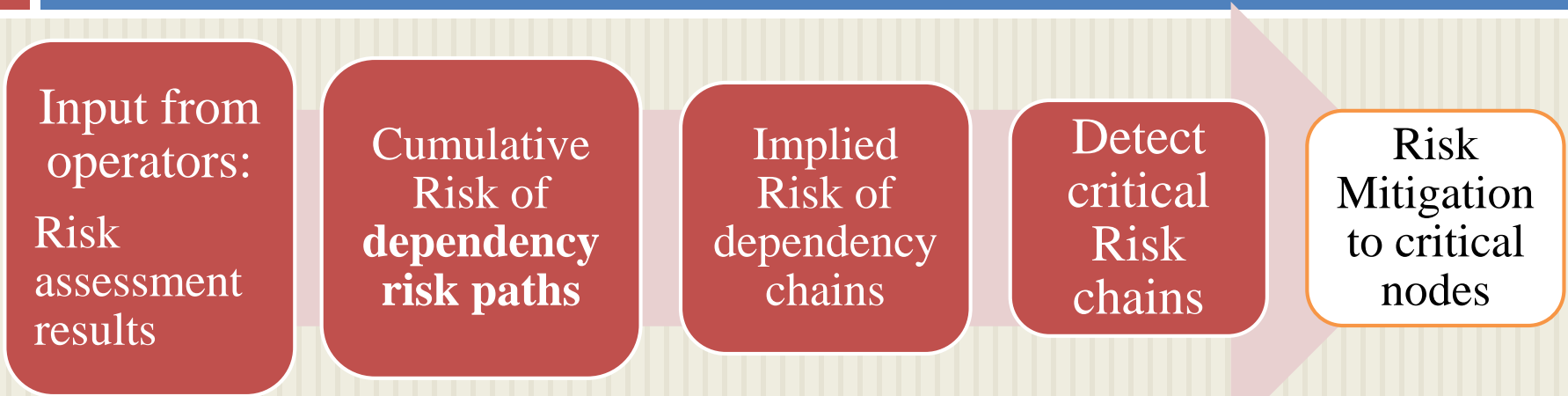
TIME-BASED IMPACT EVOLUTION IN DEPENDENCIES

9

- **RESEARCH QUESTION**: Most-Critical risk paths remain the same even if time passes?
- **SOLUTION**: Time-based analysis of Risk evolution in dependencies through time.
 - ▣ *Fuzzy Logic* analysis of impact through time.
- CI dependencies create risk paths.
 - ▣ Some paths more dangerous than others.
 - ▣ Protection of most-critical-paths a necessity.

PREVIOUS WORK ON DEPENDENCY ANALYSIS

10



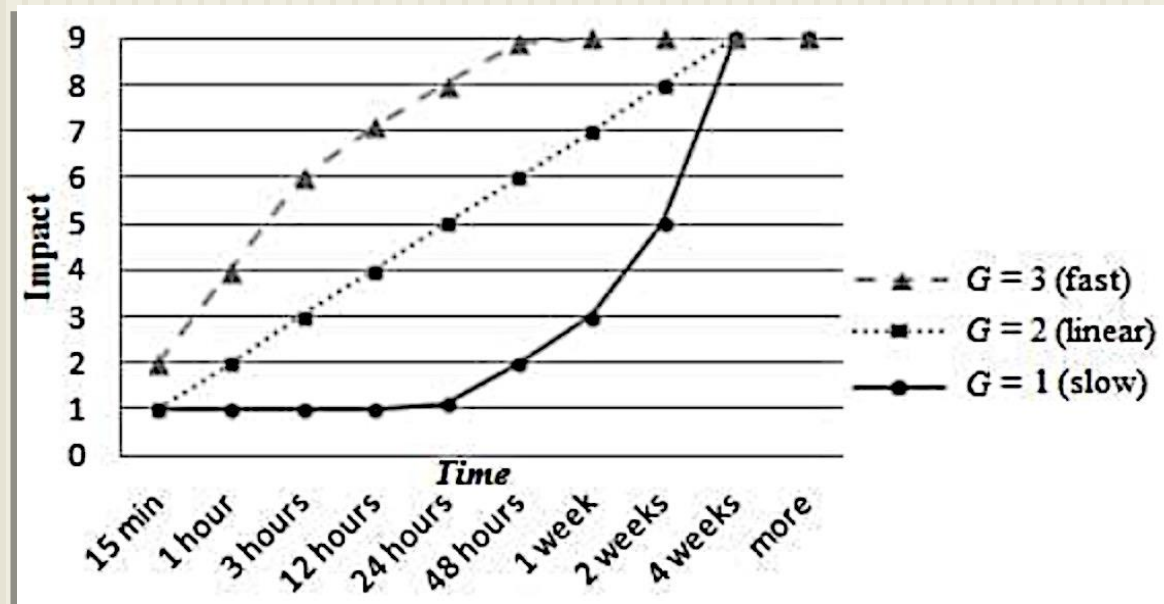
- Risk of dependency =
(**Likelihood** of occurrence) * **Impact**
- **Impact** $I_{i,j}$ and **Likelihood** $L_{i,j}$ of disruption on infrastructure node Y_j
- N-order Risk DR_{Y_0, \dots, Y_n} for a chain of disruptions:

$$DR_{Y_0, \dots, Y_n} = \sum_{i=1}^n R_{Y_0, \dots, Y_i} \equiv \sum_{i=1}^n \left(\prod_{j=1}^i L_{Y_{j-1}, Y_j} \right) \cdot I_{Y_{i-1}, Y_i}$$

TIME-BASED IMPACT EVOLUTION IN DEPENDENCIES

- Calculates dependency impact for time frames.
 - ▣ Gives estimate of impact progression from failures.
- **Worst-case** scenario and **Growth rate** of *impact* provided by risk assessments.

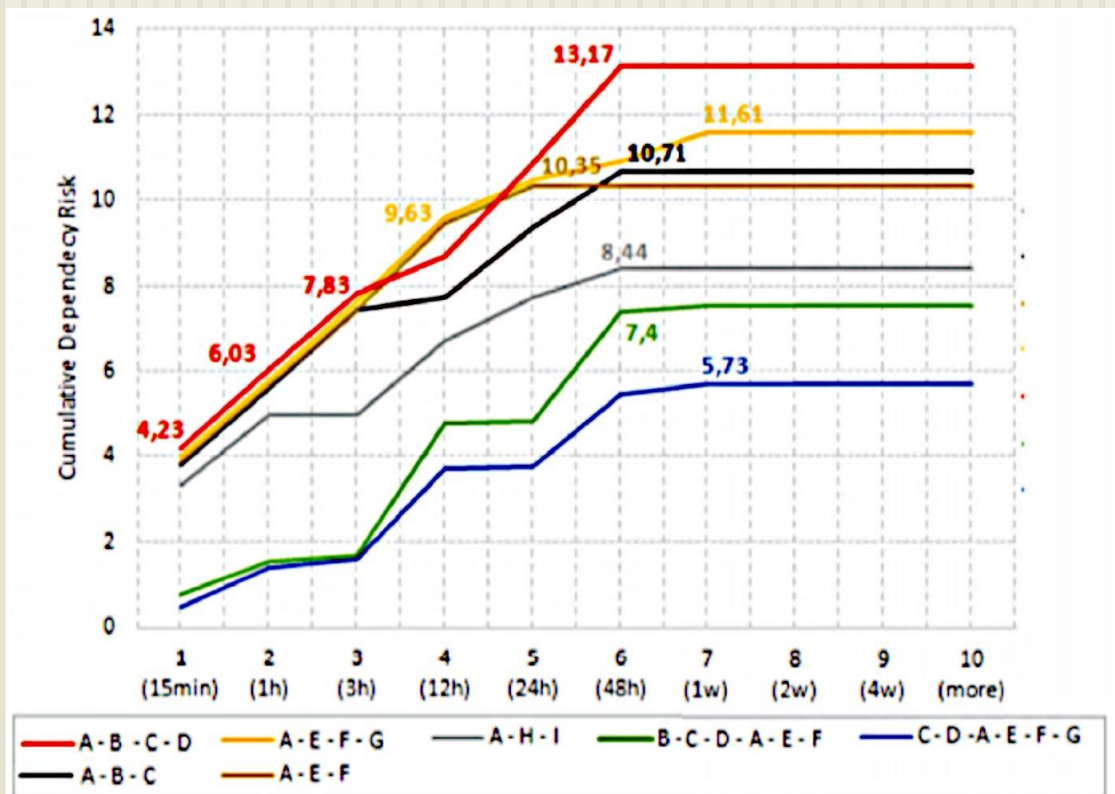
- Supports
 1. *Slow*,
 2. *Linear*
 3. *Fast*evolution of impact after failure.



TIME-BASED IMPACT EVOLUTION IN DEPENDENCIES

- **Preemptive analysis** detects dangerous paths for each time-frame.
- **Most critical CI and path** in each time slot for cascading failures.

- Different impact **growth rates** (e.g. path A-E-F-G at 3hours).

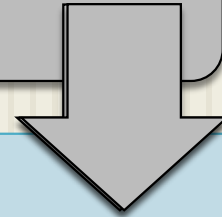


RISK MITIGATION USING GRAPH THEORY

13

1. Time-based impact evolution in dependencies

- Growth rates of failures



2. Risk mitigation using graph theory

- *Pinpoint CIs and mitigate risk*

3. Logical errors in software

- Functionality flaws lead to failures in infrastructures

RISK MITIGATION USING GRAPH THEORY

14

- **RESEARCH QUESTION**: Which nodes offer **greater overall benefit** to apply risk mitigations measures?
- **SOLUTION**: Effective risk mitigation strategies using graph Centrality metrics.
 - ▣ **Degree, Closeness, Betweenness, Eccentricity and Eigenvector** metrics.
- Key CIs affect multiple depended infrastructures.
 - ▣ Some do not belong in most critical risk paths.

RISK MITIGATION USING GRAPH THEORY

15

- **Centrality metrics** from Graph Theory to describe importance of nodes according to paths.

- Describe *importance* of interconnected infrastructure nodes in dependency Risk Graphs.
 1. *Closeness*: Average shortest path of node with others.
 2. *Eccentricity*: Greatest distance in all shortest paths.
 3. *Betweenness*: No. of paths a node participates in.
 4. *Bonacich (Eigenvector)*: Influence of node.

RISK MITIGATION USING GRAPH THEORY

- Feature selection.
- Metrics *detect dangerous CI nodes* amongst dependencies.
- ✓ 2000 Tests
- ✓ 32,950 CI nodes
- ✓ 700 graphs
- ✓ 774,015,270 paths

INFORMATION GAIN	Inbound Test	Outbound Test
Betweenness	0.259	0.277
Eccentricity	0.238	0.285
Closeness	0.387	0.345
Eigenvector	0.151	0.260
Intersection of all Centralities	0.176	0.248
Inbound degree (sinkholes)	-	0.302
Outbound degree	0.281	-

Table 1: Weka's output ranking using the Information Gain algorithm

GAIN RATIO	Inbound Test	Outbound Test
Betweenness	0.08	0.101
Eccentricity	0.08	0.101
Closeness	0.14	0.120
Eigenvector	0.06	0.09
Intersection of all Centralities	0.458	0.550
Inbound degree (sinkholes)	-	0.103
Outbound degree	0.101	-

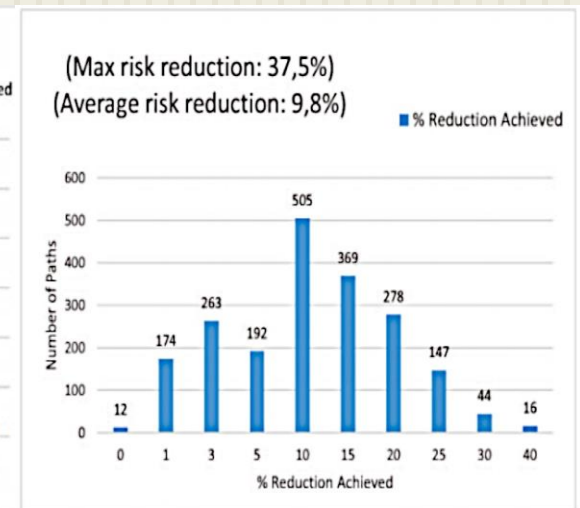
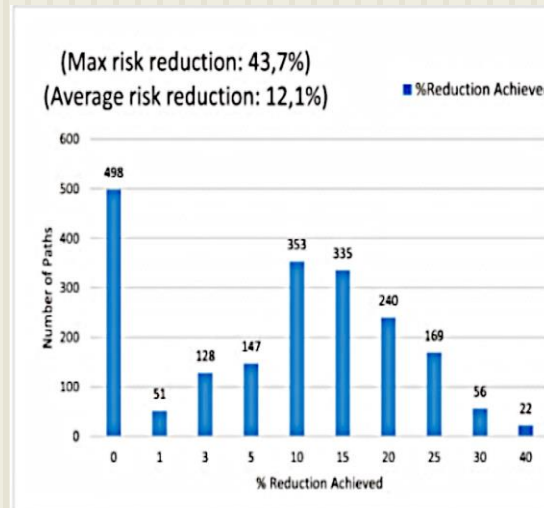
Table 2: Weka's output ranking using the Gain Ratio algorithm

RISK MITIGATION: EFFICIENCY OF SOLUTION

- Recursive selection of **CI nodes** for risk mitigation.
- Comparison** with two mitigation strategies.
- Up to **43,7%** risk reduction from algorithm.

<i>Risk Metrics</i>	<i>Strategies</i>	Information Gain	Top Initiators	Top Sinkholes
Most critical path		43.7% (max)	38.4% (max)	34.5% (max)
		12.1% (avg)	11.8% (avg)	10.3% (avg)
Top 20 critical paths		37.5% (max)	28.7% (max)	29.8% (max)
		9.8% (avg)	10.0% (avg)	7.3% (avg)
Entire graph		12.2% (max)	10.1% (max)	10.8% (max)
		7.5% (avg)	5.3% (avg)	6.7% (avg)

Table 3: Comparison of results from all mitigation strategies



LOGICAL ERRORS IN SOFTWARE

18

1. Time-based impact evolution in dependencies

- Growth rates of failures

2. Risk mitigation using graph theory

- *Pinpoint CIs and mitigate risk*

3. Logical errors in software

- Functionality flaws lead to failures in infrastructures

LOGICAL ERRORS IN FUNCTIONALITY

19

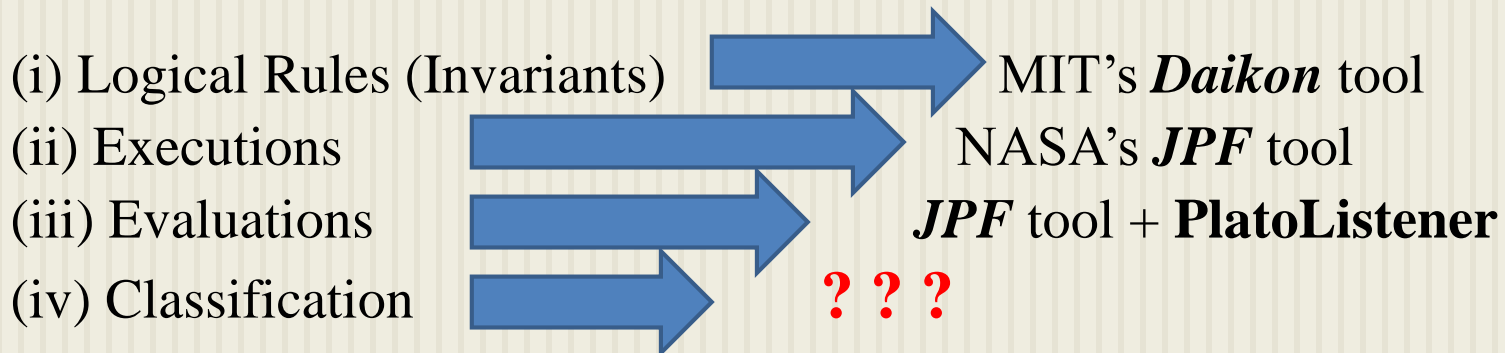
- **RESEARCH QUESTION**: Can we detect flaws in software that lead to CI failures due to erroneous implementation of business logic functionality?
- **SOLUTION**: Automated analysis of source code.
 - *Dynamic - Static analysis.*
 - *Fuzzy Logic classification.*
- Use of high-level 3rd part software to control CI functionality and services.
 - Software flaws can lead to serious failures.
 - Worst-case: Errors in functionality of applications.

LOGICAL ERRORS IN FUNCTIONALITY

20

- **Software logical errors**: erroneous translation of requirements causing unintended program behavior.

- Profiling logic behind software needs:
 - (i) **Logical rules** intended program functionality.
 - (ii) **Software Executions** with adequate coverage.
 - (iii) **Evaluations** of logical rules over executions.
 - (iv) **Classification** of detections to impact-levels.



CLASSIFICATION OF DETECTIONS

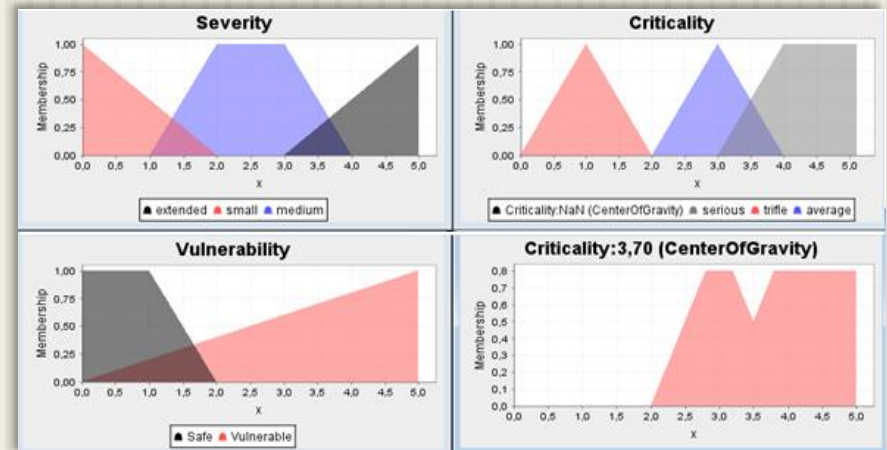
- Risk classification assigned for detections:

$$Risk(x) = Severity(x) \cap Reliability(x)$$

Reliability: *Likelihood* that a logical error exists.
Cyclomatic Density

Severity: *Impact* of error in execution flow.
Information Gain

$C = S \times R$	Low	Medium	High
Low	Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	High



LOGICAL ERRORS IN FUNCTIONALITY: METHODOLOGY

22

✓ **Plato tool** detects logical errors in high-level code.

1. Gather documentation for the software under test.

2. For each app functionality, perform dynamic analysis.

3. Infer dynamic invariants that describe functionality.

4. Instrument source code with the dynamic invariants.

5. Symbolically execute instrumented source code with JPF tool.

6. Check for violations of invariants.

7. Classify and output the Risk rank of violations.

PLATO TOOL: THREE TYPES OF EXPERIMENTS

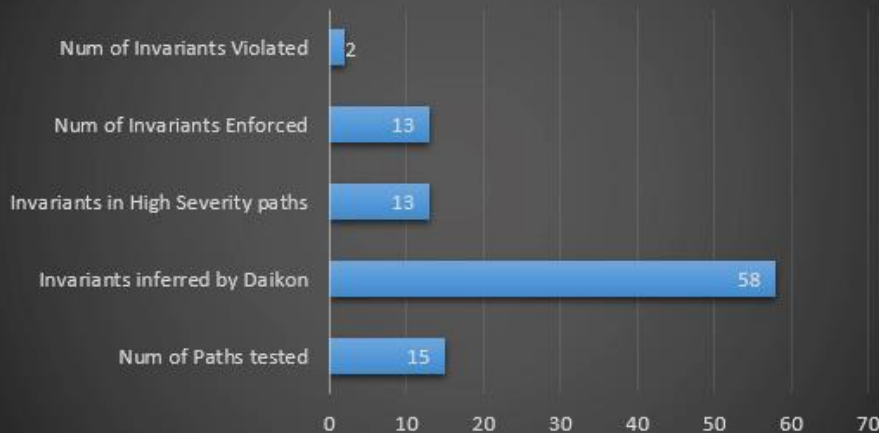
23

1. Control of RTU/PLC units in gas pipes.
2. Fault injection in NASA software.
3. Airline ticketing service.

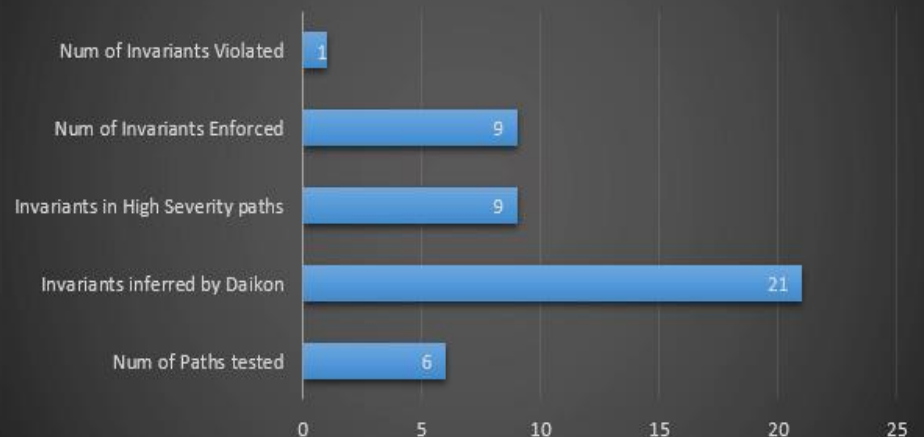
NASA's RJC - Fault Injection test



SIR object - Airline sales software



CWE - SCADA RTU software



CONCLUSIONS : *The road so far*

- Three initiatives complement each other and solve different problems.
- Different granularity level of threat analysis.
 - ✓ Impact evolution detects extra high-risk paths-per-time frame.
 - ✓ Detect dangerous CI nodes that affect multiple dependencies.
 - ✓ Efficient risk mitigation strategies tried & tested.
 - ✓ Detect logical errors in software that handles functionality.
 - ✓ Predict failures to avoid catastrophic failures.

Time-based analysis – I(t) tables

25

- ✓ Calculation of all possible I(t) values
- ✓ Pre-computed table - expected time-related impact values for fast evolving failures
- ✓ Worst impact at T = 12 hours

		Time Related Impact									
		Very Low	Low	Low	Medium	Medium	High	High	Very High	Very High	
		1	2	3	4	5	6	7	8	9	
Early	15 minutes	1	1	2	2	2	3	3	3	4	
	1 hour	1	2	2	3	3	4	4	5	6	
	3 hours	1	2	3	3	4	5	5	6	7	
Medium	12 hours	1	2	3	4	5	6	7	8	9	
	24 hours	1	2	3	4	5	6	7	8	9	
	48 hours	1	2	3	4	5	6	7	8	9	
Late	1 week	1	2	3	4	5	6	7	8	9	
	2 weeks	1	2	3	4	5	6	7	8	9	
V.Late	4 weeks	1	2	3	4	5	6	7	8	9	
	> 4 weeks	1	2	3	4	5	6	7	8	9	

```
foreach e2 in set T do
```

```
  Set e2 as t;
```

```
  /* For t > T always output maximum impact
```

```
  if t > T then
```

```
    | I(t) = I;
```

```
  else if t ≤ T then
```

```
    foreach I in set I do
```

```
      if G is fast then
```

```
        | I(t) = I · logT t;
```

```
      else if G is linear then
```

```
        | I(t) = I · (t/T);
```

```
      else if G is slow then
```

```
        | I(t) = I(t/T);
```

```
    end
```

```
end
```

TIME-BASED ANALYSIS: EXAMPLE

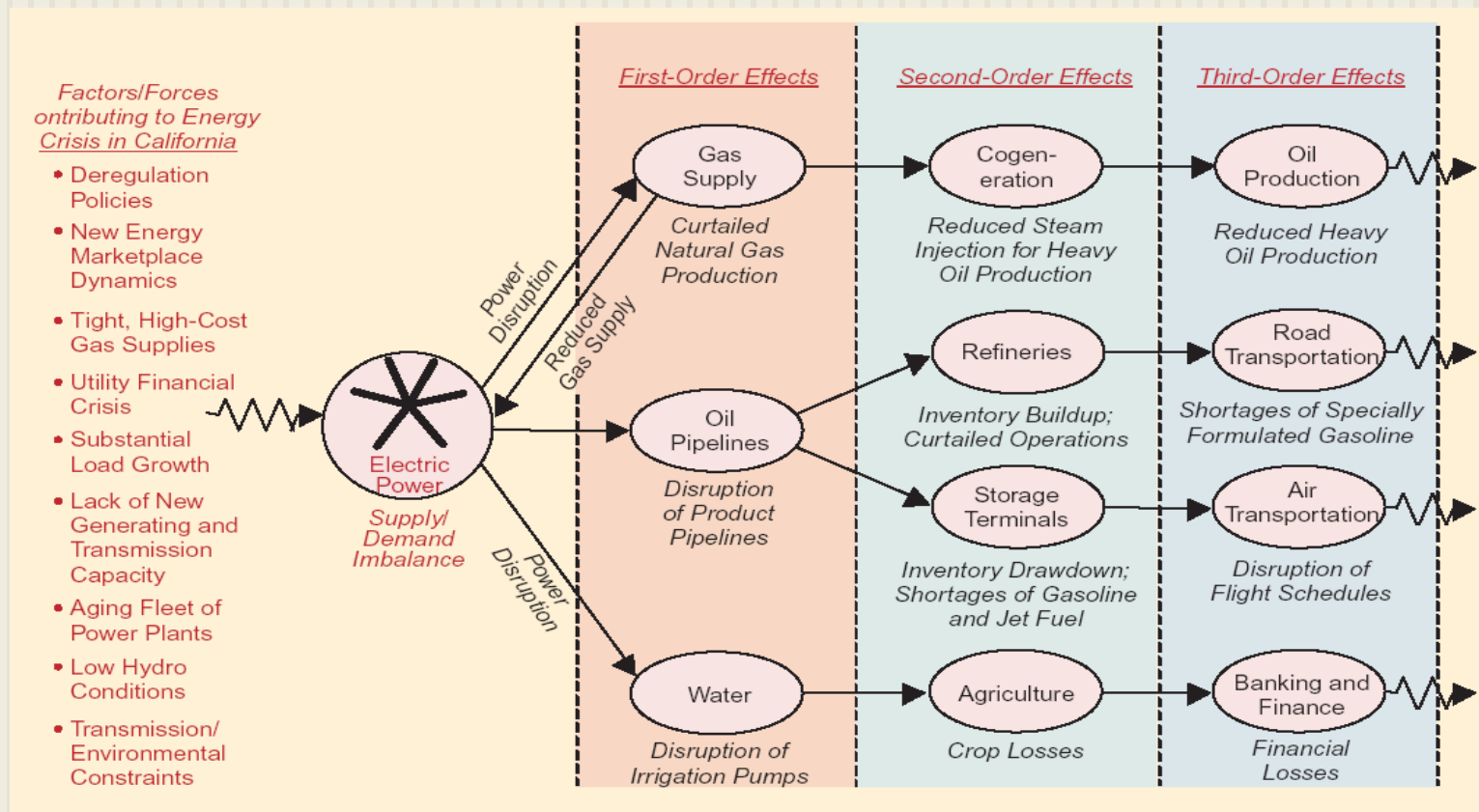
26

- **Example:** Examined dependency input data $G = \text{fast}$ and $T = 12\text{h}$.
- 1. Fuzzy membership set Low membership percentages: $\text{Low} = \{(1, 0.05) (2, 0.55) (3, 0.4)\}$ where the second value in each couple is the membership percentage of the corresponding impact value.
- 2. Subset of the rules used to calculate the Low output set of the values $I(t)$ are the following:
 - 17: IF Impact IS Low AND Time IS Early THEN Fuzzy Impact is Very Low;
 - 18: IF Impact IS Low AND Time IS Medium THEN Fuzzy Impact is Medium;
 - 19: IF Impact IS Low AND Time IS Late THEN Fuzzy Impact is High;
 - 20: IF Impact IS Low AND Time IS Very Late THEN Fuzzy Impact is High;
- 1. Fuzzy set mostly characterized by that value is the Low set.
- 2. Based on Table 2, this time belongs to the Medium time fuzzy set.
 - Thus, using rule 18, CIDA's setup process will choose to output a Medium Fuzzy Impact value.
 - RightMostMembership defuzzification technique on all impact value.

TIME-BASED ANALYSIS: CALIFORNIA BLACK OUT SCENARIO

27

- ✓ California Blackout Scenario used to test the methodology implemented CIDA tool



PROPOSED ALGORITHM – CENTRALITY FEATURES

1. *Assess the cumulative dependency risk of all existing dependency paths*
2. *Compute all centrality measures for every node.*
3. *Select a subset of nodes for applying risk mitigation controls, based on centrality measures.*
 - A. *Calculate Information Gain of each centrality measure for selected subset of nodes.*
 - B. *Recurse by choosing highest output and reapplying calculations in new subset.*
4. *Evaluate results by comparing new graph to initial one (risk of most critical path, max risk of all paths, or no. of paths with high risk).*

DECISION-TREE TRAVERSAL USING INFORMATION GAIN

29

✓ Selects nodes for risk mitigation

Procedure choosingNodeSubsets ($C, IG_c(D), N, D$)

Inputs

High centrality subsets: $C = \{C_C, C_I, C_B, C_{Eg}, C_E\}$

Head set of all nodes: N

Number of nodes to be chosen from D : $PERCENT$

Subset of dangerous nodes: $D \subset N$

Information gain of a centrality subset c over D : $IG_c(D)$

Output

Percentage of nodes from $D' \subset N$ for risk mitigation

Step 1 – Start:

Mark all sets in C as unused

Position at the root node

Step 2 – Leaf:

If C is empty then

 Output top nodes from D with highest centrality
 END

else

 if $IG_c(D) = 0 \forall c \in C$ then

 Output top nodes from D with highest centrality
 END

 end if

end if

Step 3 – Decide case:

if nodes in $D < PERCENT$ then

 Go to Step 4

else

 Go to Step 5

end if

Step 4 – Backtrack:

if at the root then

 STOP

else

 Return to the node above the current node

choose different $c \in C$

 Go to Step 3

end if

Step 5 – Decision:

if $IG_{c_1}(D) = IG_{c_2}(D) \forall c_1, c_2 \in C$ then

 Select the leftmost c from C

else

 Calculate information gain $IG_c(D) \forall c \in C$
 Select c from C with the highest $IG_c(D)$

end if

$N \leftarrow N \setminus c$

$D \leftarrow D \setminus c$

$C \leftarrow C - \{c\}$

Go to Step 2

end procedure

FORMAL DEFINITION OF LOGICAL ERRORS

30

Definition 1. A *logical error* manifests if there are execution paths π_i and π_j with the same prefix, such that for some $k \geq 0$ the transition $(\ell_k, \rho_k, \ell_{k+1})$ results in states $(\ell_{k+1}, s_i), (\ell_{k+1}, s_j)$ with $s_i \neq s_j$ and for the dynamic invariant r_k , $(s_{i-1}, s_i) \models r_k$ in π_i and $(s_{j-1}, s_j) \not\models r_k$ in π_j , i.e. r_k is satisfied in π_i and is violated in π_j .

So, for each dynamic invariant r_k , PlatoListener relies on JPF's symbolic execution to gather all execution paths that evaluate the dynamic invariant. Then, for each path with some state s_j such that $(s_{j-1}, s_j) \not\models r_k$, PLATO compares it with other paths with the same prefix such that $s_i \neq s_j$ and $(s_{i-1}, s_i) \models r_k$.

DETECTING ERRORS THAT CAUSE FAILURES. WHAT WE NEED?

31

VARIABLE: ix_31 <- 0

VARIABLE: ix_31 -> 0

rjc.SimpleCounter.Main([I]V

VARIABLE: execute_at_initialization_933 -> 1

rjc.SimpleCounter.SimpleCounter_100000256_exec
()V

rjc.SimpleCounter.CounterState_100000257_exec()
V

VARIABLE:

TopLevel_SimpleCounter_count -> 0

VARIABLE:

TopLevel_SimpleCounter_count <- 1

rjc.Chart.Chart_100000202_exec():::
ENTER

this.TopLevel_Chart_count == 2.0

this.TopLevel_Chart_Firefct1 ==
-0.005235987755982988

this.TopLevel_Chart_Firefct1 ==
this.TopLevel_Chart_Coastfct1

JPF - Static analysis
Produces method execution paths

Daikon - Dynamic analysis Produces
invariant rules

SEVERITY CLASSIFICATION: INFORMATION GAIN ALGORITHM

32

- Severity classification based on Expected Information Gain measure.
 - ▣ Successful in feature selection for information retrieval
- Uses taxonomy of dangerous code instructions.
 - ▣ Instructions tied to known vulnerability types
 - ▣ 5 subsets/danger levels of instructions - Feature Sets to classify execution paths.
- Severity ratings applied by classifying each execution path into a Severity set

Rank	Example of classified methods	Feature Set (Category level)
Low	javax.servlet.http.Cookie	Set 1 (Level 1)
Low	java.lang.reflection.Field	Set 2 (Level 2)
Medium	java.io.PipedInputStream	Set 3 (Level 3)
High	java.io.FileInputStream	Set 4 (Level 4)
High	java.sql.ResultSet::getString	Set 5 (Level 5)

SEVERITY CLASSIFICATION: TAXONOMY

33

- Instructions assigned to feature sets, i.e. danger levels.
- Correct feature set inferred based on the CVSS scores in NVD repository.
- Implemented using the following algorithm:
 1. For each instruction, check lowest and highest ratings of NVD vulnerabilities that use this instruction.
 2. Characteristics inputted in CVSS 3.0 scoring calculator 2.
 - ▣ *Calculate lowest and highest possible vulnerability scores.*
 3. Instructions with score 7 or above grouped in Set 5. Instructions with score 6 to 7 in Set 4, those with score 5 to 6 in Set 3, those with score 4 to 5 in Set 2 and those with score 1 to 4 in Set 1.

RELIABILITY CLASSIFICATION: CYCLOMATIC DENSITY ALGORITHM

34

- Cyclomatic Complexity doesn't consider size of code.
- Modules with both high complexity and large size tend to have the lowest reliability.

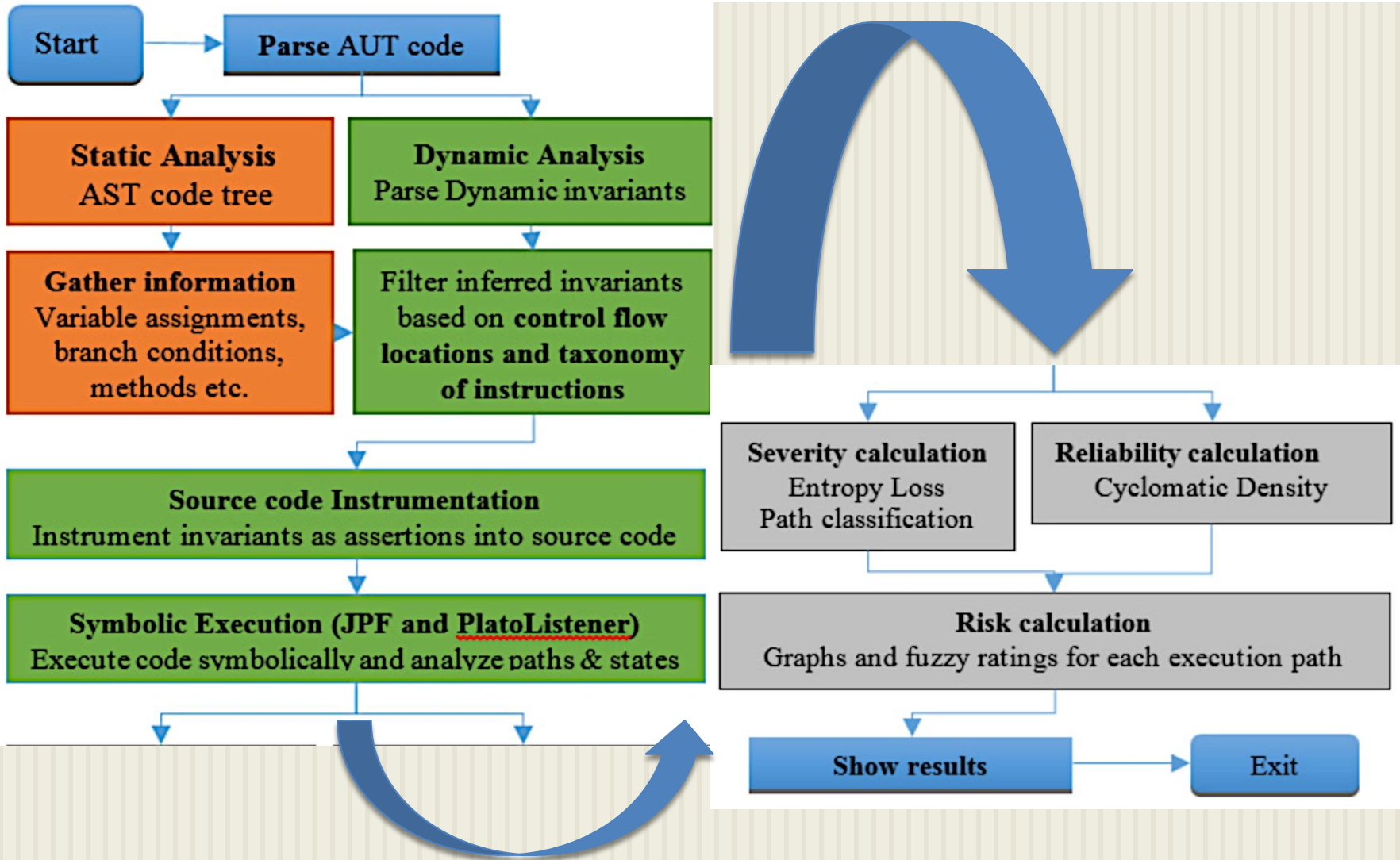
Compute V(G):

1. Increment one for every IF, CASE or other same construct;
2. Increment one for every DO, DO-WHILE or other same construct;
3. Add two less than number of logical alternatives in CASE;
4. Add one for each logical operator (AND, OR) in an IF.

Rank	Example of classified methods	Category
Low	Cycl. Complexity Density ≤ 0.1	1
Low	Cycl. Complexity Density >0.1 && Cycl. Complexity Density ≤ 0.2	2
Medium	Cycl. Complexity Density >0.2 && Cycl. Complexity Density ≤ 0.3	3
High	Cycl. Complexity Density >0.3 && Cycl. Complexity Density ≤ 0.4	4
High	Cycl. Complexity Density >0.4	5

PLATO: PROCESSING FLOWCHART

35



TEST RESULTS: NASA'S RJC CONTROLLER

36

- Error injection to source code with high representativeness
 - Code Metrics: Lines of Code, Cyclomatic Complexity, Average methods per Class.
 - Detect key points in the source code for fault injection.
- Evaluation of system behavior when one of its components is faulty and not the behavior of the faulty component itself.

	Lines of code	Cyclomatic complexity	Average methods/type
Rjc.Chart.java	10,48	3,31	29
Rjc.Chart_1.java	13,68	3,31	29
Rjc.Chart_2.java	13,68	3,31	29
Rjc.Reaction_Jet_Control0.java	99,50	7,50	2
Rjc.Reaction_Jet_Control1.java	85,50	7,50	2

- PLATO's Fuzzy Logic system classified violations with the following ratings:
 - Severity = 2,
 - Reliability = 1.

TEST RESULTS: SOFTWARE-TO-RTU/PLC CONTROLS

37

Step 1. Functionality has two flows currently available:

- A. Exec choice(1), Exec choice(2).
- B. Exec choice(2), Exec choice(1), Exec choice(2).

Steps 2-3. Dynamic analysis: 40 invariants for selected functionality. Next invariant refers to hidden logical error:

```
Bug.readRegisterPressure()::ENTER  
    this.checked == false
```

Steps 4-5. Invariants instrumented in source code points.

- ▣ Software was executed symbolically.

Step 6-7. Assertion violation detected - readRegisterPressure(): Two executions where variable was TRUE and FALSE respectively, implying a logical error.

- ▣ **Severity = 3** and **Vulnerability = 5** yielding a **Risk value of ~4** for detection.

TEST RESULTS: AIRLINE TICKETING SERVICE – SIR OBJECT

38

Step 1-3. There is only one function point to test. Daikon inferred the following invariant amongst others:

Num Of Seats Sold \leq this.Maximum Capacity

Step 4-5. Dynamic invariants instrumented in the source code and the software was symbolically executed in JPF.

Step 6-7. Assertion violation detected - runBug(): two executions were found where the mentioned invariant was enforced and violated respectively, thus implying a possible logical error.

- ▣ **Severity** = 5 score and a **Reliability** = 3, yielding a Risk value of 4.5

References

1. Stergiopoulos G., Kotzanikolaou P., Theoharidou M., Gritzalis D., "Risk mitigation strategies for Critical Infrastructures based on graph centrality analysis", *International Journal of Critical Infrastructure Protection*, September 2015.
2. Kotzanikolaou P., Theoharidou M., Gritzalis D., "Accessing n-order dependencies between critical infrastructures", *International Journal of Critical Infrastructures*, Vol. 9, Nos. 1-2, pp. 93-110, 2013.
3. Theoharidou M., Kotzanikolaou P., Gritzalis D., "Risk assessment methodology for interdependent critical infrastructures", *International Journal of Risk Assessment and Management (Special Issue on Risk Analysis of Critical Infrastructures)*, Vol. 15, Nos. 2/3, pp. 128-148, 2011.
4. Theoharidou M., Kotzanikolaou P., Gritzalis D., "A multi-layer Criticality Assessment methodology based on interdependencies", *Computers & Security*, Vol. 29, No. 6, pp. 643-658, 2010.
5. Theoharidou M., Xidara D., Gritzalis D., "A Common Body of Knowledge for Information Security and Critical Information and Communication Infrastructure Protection", *International Journal of Critical Infrastructure Protection*, Vol. 1, No. 1, pp. 81-96, 2008.
6. Kotzanikolaou P., Theoharidou M., Gritzalis D., "Cascading effects of common-cause failures on Critical Infrastructures, in Proc. of the 7th IFIP International Conference on Critical Infrastructure Protection (CIP-2013), pp. 171-182, Springer (AICT 417), USA, March 2013.
7. Kotzanikolaou P., Theoharidou M., Gritzalis D., "Interdependencies between Critical Infrastructures: Analyzing the Risk of Cascading Effects", in Proc. of the 6th International Conference on Critical Infrastructure Security (CRITIS-2011), pp. 104-115, Springer (LNCS 6983), 2013.
8. Stergiopoulos G., Petsanas P., Katsaros P., Gritzalis D., "Automated exploit detection using path profiling: The disposition should matter, not the position", in Proc. of the 12th International Conference on Security and Cryptography (SECRYPT-2015), pp. 100-111, ScitePress, France, July 2015.
9. Stergiopoulos G., Kandias M., Gritzalis D., "Approaching Encryption through Complex Number Logarithms" (position paper), in Proc. of the 10th International Conference on Security and Cryptography (SECRYPT-2013), pp. 574-579, Samarati P., et al. (Eds.), Iceland, July 2013.
10. Polemi D., Ntouskas T., Georgakakis E., Douligeris C., Theoharidou M., Gritzalis D., "S-Port: Collaborative security management of Port Information Systems", in Proc. of the 4th International Conference on Information, Intelligence, Systems and Applications (IISA-2013), IEEE Press, Greece, July 2013.
11. Stergiopoulos G., Tsoumas V., Gritzalis D., "On Business Logic Vulnerabilities Hunting: The APP_LogGIC Framework", in Proc. of the 7th International Conference on Network and System Security (NSS 2013), pp. 236-249, Springer (LNCS 7873), Spain, June 2013.
12. Stergiopoulos G., Tsoumas B., Gritzalis D., "Hunting application-level logical errors", in Proc. of the 4th International Symposium on Engineering Secure Software and Systems (ESSOS-2012), Livshits B., et al (Eds.), pp. 135-142, Springer (LNCS 7159), The Netherlands, February 2012.
13. Theoharidou M., Kandias M., Gritzalis D., "Securing Transportation-Critical Infrastructures: Trends and Perspectives", in Proc. of the 7th IEEE International Conference in Global Security, Safety and Sustainability (ICGS3-2011), pp. 171-178, Springer (LNICST 0099), Greece, 2012.

14. Stergiopoulos G., Kotzanikolaou P., Theoharidou M., Gritzalis D., “Using centrality metrics in CI dependency risk graphs for efficient risk mitigation”, in Proc. of the 9th IFIP International Conference on Critical Infrastructure Protection (CIP-2015), Springer, USA, March 2015.
15. Stergiopoulos G., Theoharidou M., Gritzalis D., "Using logical error detection in Remote-Terminal Units to predict initiating events of Critical Infrastructures failures", in Proc. of the 3rd International Conference on Human Aspects of Information Security, Privacy and Trust (HCI-2015), Springer, USA, August 2015.
16. Stergiopoulos G., Katsaros P., Gritzalis D., “Automated detection of logical errors in programs”, in Proc. of the 9th International Conference on Risks and Security of Internet and Systems (CRiSIS-2014), Springer, August 2014.
17. Stergiopoulos G., Katsaros P., Gritzalis D., “Source code profiling and classification for automated detection of logical errors”, in Proc. of the 3rd International Seminar on Program Verification, Automated Debugging and Symbolic Computation (PAS-2014), Austria, July 2014.
18. Soupionis Y., Benoist T., “Cyber attacks in Power Grid ICT systems leading to financial disturbance”, in Proc. of the 9th International Conference on Critical Information Infrastructures Security (CRITIS-2014), Springer, Cyprus, October 2014.
19. Virvilis N., Gritzalis D., “Trusted Computing vs. Advanced Persistent Threats: Can a defender win this game?”, in Proc. of 10th IEEE International Conference on Autonomic and Trusted Computing (ATC-2013), pp. 396-403, IEEE Press, Italy, December 2013.
20. Theoharidou M., Papanikolaou N., Pearson S., Gritzalis D., “Privacy risks, security and accountability in the Cloud”, in Proc. of the 5th IEEE Conference on Cloud Computing Technology and Science (CloudCom-2013), pp.177-184, IEEE Press, United Kingdom, December 2013.
21. Virvilis N., Gritzalis D., “The Big Four - What we did wrong in Advanced Persistent Threat detection?”, in Proc. of the 8th International Conference on Availability, Reliability and Security (ARES-2013), pp. 248-254, IEEE, Germany, September 2013.
22. Theoharidou M., Kotzanikolaou P., Gritzalis D., "Towards a Criticality Analysis Methodology: Redefining Risk Analysis for Critical Infrastructure Protection", in Proc. of the 3rd IFIP International Conference on Critical Infrastructure Protection (CIP-2009), C. Palmer, S. Shenoj (Eds.), Springer, USA, March 2009.