Reducing Software Security Risk Through an Integrated Approach

David Gilliam, John Powell, Josef Sherif, & Tom Wolfe
California Institute of Technology, Jet Propulsion Laboratory

Matt Bishop
University of California at Davis

NASA Center Initiative: Reducing Software Security Risk

- **NOTE:**
  - **This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.**
  - **The work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program lead by the NASA Software IV&V Facility.**
  - **This activity is managed locally at JPL through the Assurance and Technology Program Office.**

Current Collaborators

- David Gilliam – Principle Investigator
  Network and Computer Security, JPL
- John Powell – Research Engineer
  Quality Assurance, JPL
- Josef Sherif – Research Engineer
  Network and Computer Security, JPL
- Tom Wolfe – Research Engineer
  Network and Computer Security, JPL
- Matt Bishop –Professor of Computer Science
  University of California at Davis

Introduction

- Research Goal
- Previously Reported Work
- Software Security Checklist
- Integration of the instrument contents
- The Flexible Modeling Framework (FMF)
  - FMF components and combinations
  - Rapid model updating
- Summary
- Future Work

Research Goal

- Reduce security risk to the computing environment by preventing and/or mitigating vulnerabilities and exposures through the software life cycle process
  - Develop a Software Security Assessment Instrument (SSAI)
  - Five foci
    - Software Security Checklist (SSC)
    - Model-based security specification and validation approach (MBV)
      - Flexible Modeling Framework Instrument (FMF)
    - Vulnerability Matrix (VMatrix)
    - Software property-based testing tool (PBT)
    - List of Security Assessment Tools (SAT)

Previously Reported Components

- Vulnerability Matrix
  - Matrix listing vulnerabilities and exploit against the vulnerability
- Security Assessment Tools (SATs)
  - List of tools that can be used for assessing security of code, their use and alternative tools (Insure++, RAT4, etc.)
- Property-Based Testing
  - Tester's Assistant – uses a technique for testing that programs meet given specifications
  - Code slicing using assertions (changes in security state of program) and properties (describe specific security state)
  - JAVA source code

Software Security Checklist (SSC)

- NASA NPG 2210
  - External Release of Software
  - Software Security Checklist
- Two phases
  - Phase 1: Provide instrument to verify external release of NASA software does not present a security risk to NASA and its partners
  - Phase 2: Provide instrument to integrate security as a formal approach to the software life cycle

Software Security Checklist (cont.)

- Phase 1
  - Checklist for use prior to external release of software
    - Concerns include the following:
      - HR information
      - Operational processes
      - Internal processes
      - Sensitive IP addresses, Host names, etc.
      - Crypto Certificates and Keys
      - Embedded userids and passwords
    - Document describing purpose and use of the checklist

Software Security Checklist (cont.)

- Proposed procedure for using the checklist
- Examples for automating (as much as possible) software checking
  - Perl script for checking strings with IP addresses
  - Perl script for checking dangerous subroutines
    - gets, fgets, strcpy, strcat, sprintf, printf, and fprintf, because of format string errors
    - strncpy and strncat because of the oft-forgotten NUL byte problem, and possibly others
  - The specific set of functions is system dependent; for example, some Windows functions require parameter checks that UNIX versions would not, i.e., "system" is not a function that works on Windows

Software Security Checklist (cont.)

- Phase 2
  - Determine scope of Phase 2
    - Analyze and prioritize what items are of concern
      - Cost of life cycle (Risk cost grows exponentially throughout software life cycle)
      - Cost of where problems occur
      - Security risk analysis
    - Determine and prioritize most effective solutions for addressing the concerns
      - Need for software security for architect?

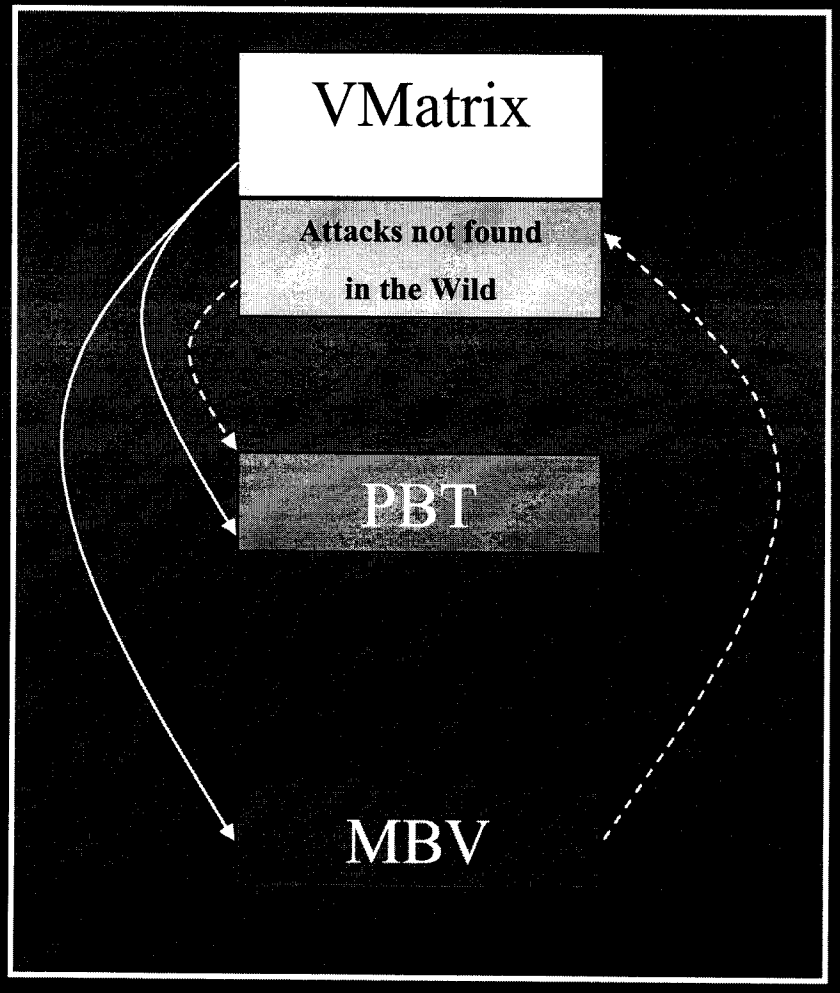
Software Security Checklist (cont.)

- Develop Checklists for use through the software life cycle process
- Develop Document describing purpose and use of the checklists
- Develop or Provide list of Tools/Instruments that can assist in automation of checks
  - these tools help a good analyst, but they cannot replace him or her (Halting problem, etc.)
- Provide Recommendations and lessons learned for software security architecture

Instrument Contents Integration

Each part of the instrument supports the other parts

- VMatrix provides known vulnerability properties to PBT
- PBT provides newly discovered code vulnerabilities to the Vmatrix property set and code level verification feedback to MBV
- MBV provides newly discovered vulnerabilities scenarios to the Vmatrix property set and early lifecycle verification results to PBT for tracability purposes



Software Security Checklist (cont.)

- Phase 2
  - Determine scope of Phase 2
    - Analyze and prioritize what items are of concern
      - Cost of life cycle (Risk cost grows exponentially throughout software life cycle)
      - Cost of where problems occur
      - Security risk analysis
    - Determine and prioritize most effective solutions for addressing the concerns
      - Need for software security for architect?

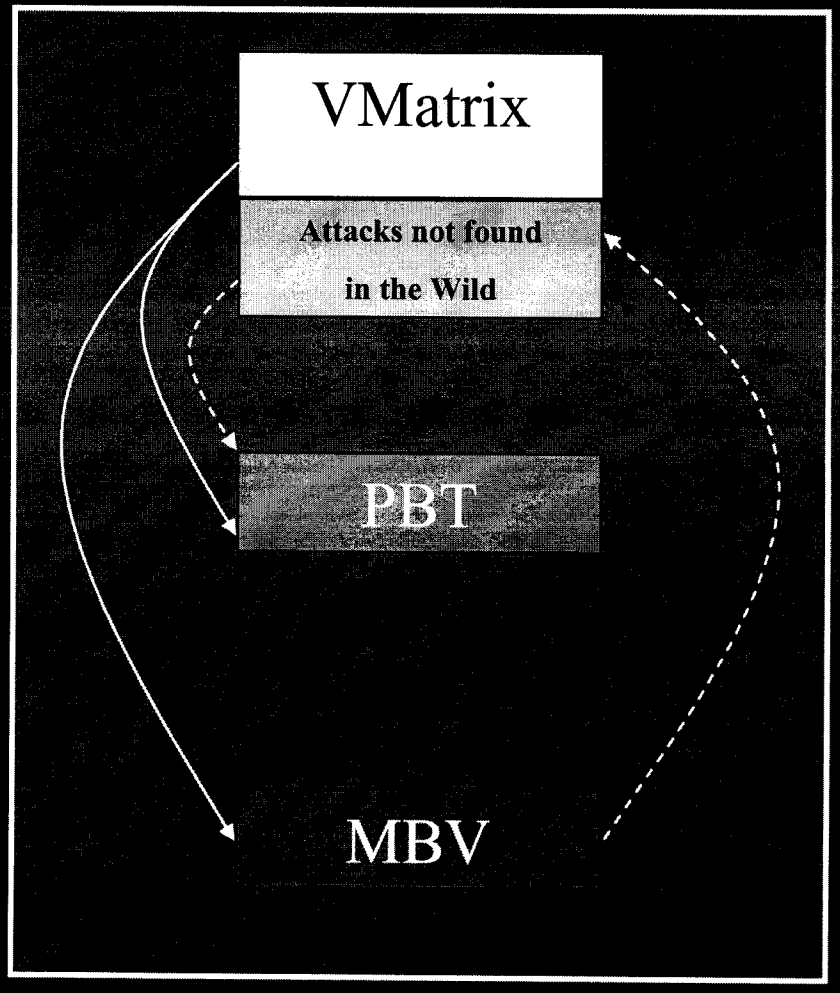
Software Security Checklist (cont.)

- Develop Checklists for use through the software life cycle process
- Develop Document describing purpose and use of the checklists
- Develop or Provide list of Tools/Instruments that can assist in automation of checks
  - these tools help a good analyst, but they cannot replace him or her (Halting problem, etc.)
- Provide Recommendations and lessons learned for software security architecture

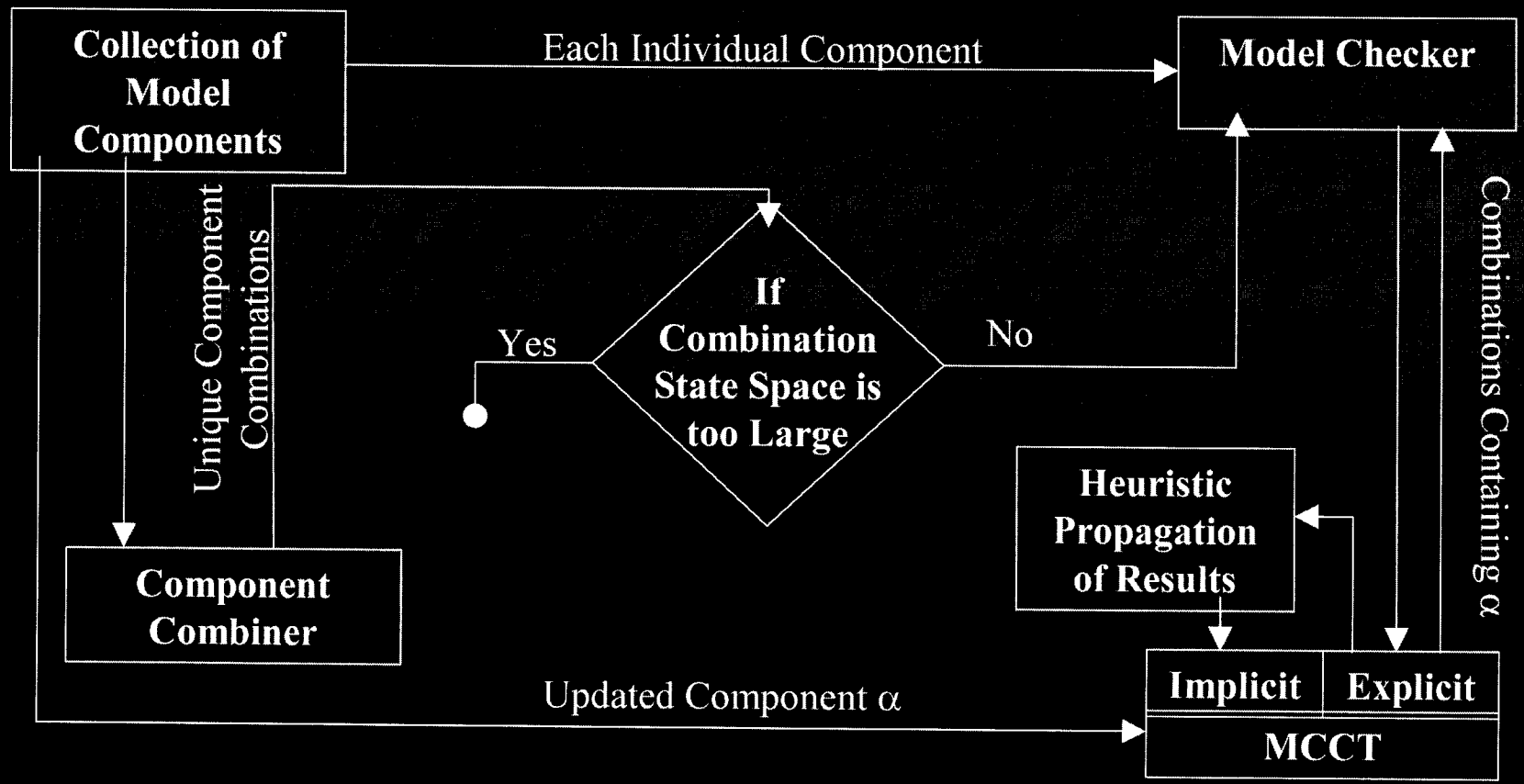
Instrument Contents Integration

Each part of the instrument supports the other parts

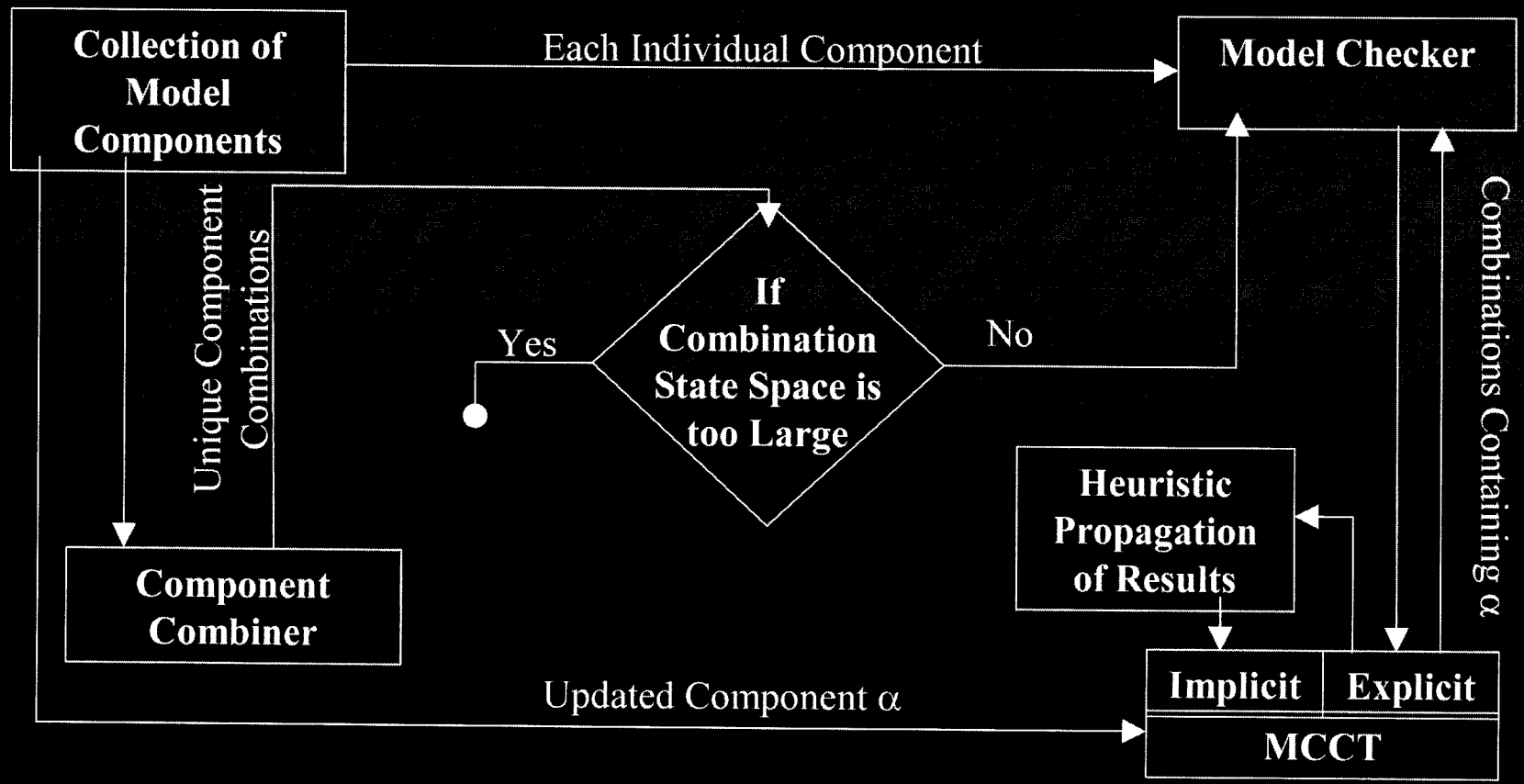
- VMatrix provides known vulnerability properties to PBT
- PBT provides newly discovered code vulnerabilities to the Vmatrix property set and code level verification feedback to MBV
- MBV provides newly discovered vulnerabilities scenarios to the Vmatrix property set and early lifecycle verification results to PBT for tracability purposes



The Flexible Modeling Framework (FMF)



The Flexible Modeling Framework (FMF)



Model Checking (MC)

- FMF uses MC as its core technology
- Exhaustive search
- Verification of temporal logic properties
- MC limited by an exponential state space explosion

  m^n s.t. m=val_range, n=|vars|

Can MC still be useful when the system is large?

FMF Components and Combinations

- FMF models the system (S) as a series of small model components ($c_i \mid 1 \leq i \leq n$) which are model checked

  $$(\{c_1, c_2, c_3, \ldots c_n\} = S) \wedge (c_i \subset S)$$

- FMF model checks component combinations

  $(C_k \mid (C_k \subset S) \wedge (1 \leq |C_k| \leq m)$ where the state space of C_{m+1} is beyond memory thresholds

Model Component Combination Tree (MCCT)

- MCCT level l = $\{C_k \mid (|C_k| = n-l) \wedge (n = |S|)\}$
  - MCCT root node (level 0) – full system of n components
  - MCCT leaf nodes (level n-(n-1)) – single components
- State space explosion threshold separates the explicit and implicit portions of the MCCT
  - Explicit portion – **<C_k, VV = 0||2, CR = 1>**
    - known property verification knowledge from MC results
  - Implicit portion – **<C_k, VV = (0,2), CR = (0,1)>**
    - derived verification knowledge from explicit portion of MCCT

A Simple Example

$CR_{i+1} = CR_i - p \rightarrow$ $VV_{ABCD} = 1.33$

Implicit MCCT

$0 < CR < 1$

$CR_i = 1 - p \rightarrow$

$VV_{ABC} = 1.33$ $VV_{ABD} = 2.00$ $VV_{ACD} = 1.33$ $VV_{BCD} = 0.67$

$VV_{AB} = 2$ $VV_{AC} = 2$ $VV_{AD} = 2$ $VV_{BC} = 0$ $VV_{BD} = 2$ $VV_{CD} = 0$

$CR = 1 \rightarrow$

$CR = 1 \rightarrow$ $VV_A = 0$ $VV_B = 2$ $VV_C = 2$ $VV_D = 2$

Explicit MCCT

$CR = 1$

Potential Follow-On Work

- Provide training in use of security assessment tools in the software development and maintenance life-cycle
- Develop capability for easy storage and access of a library of common network security model components and past verification results
- Develop a programmer interface to assist users with generating properties for input into the tools

Potential Follow-On Work (cont.)

- Enhancing and augmenting the toolset
  - Port the code to run on different operating systems in a run-time environment
  - Include additional programming and scripting languages that the Tester's Assistant tool can slice for vulnerabilities
  - Augment the toolset by incorporating or developing additional tools (SATs)
  - Develop a graphical user interface front-end checklist and decision tree to assist in building the Model to be verified
  - Develop an interface into the AART Tool

Summary

- Growth of NASA's network aware software applications and collaborative work increase risk to NASA environment
  - Risk will continue to increase as collaboration increases
- Software Security Assessment Instrument for use in the software development and maintenance lifecycle

Summary (Cont.)

- Assessment Instrument composed of four tools and reports:
  - Vulnerability Matrix (VMatrix)
  - Tester's Assistant (PBT)
  - Flexible Modeling Framework (MBV, FMF)
  - Software Security Checklist (SSC)
- Tools can be used collectively or individually
- There is a potential for wider application of the instrument beyond assessment of security of software

FOR MORE INFO...

David Gilliam
JPL
400 Oak Grove Dr., MS 144-210
Pasadena, CA 91109
Phone: (818) 354-0900 FAX: (818) 393-1377
Email: david.p.gilliam@jpl.nasa.gov
Website: http://security.jpl.nasa.gov/rssr/

John Powell
MS 125-233
Phone: (818) 393-1377
Email: john.d.powell@jpl.nasa.gov

Backup Slides

JPL Network and Computer Security Group

CL01-1551

Reducing Software Security Risk Through an Integrated Approach (RSSR) RTOP

Research Project Description:

This research project is joint work by the California Institute of Technology's Jet Propulsion Laboratory and the University of California at Davis sponsored by the National Aeronautics and Space Administration to develop a security assessment instrument for the software development and maintenance life cycle. The assessment instrument is a collection of tools and procedures to support development of secure software.

The assessment instrument consists of a Vulnerability Matrix (VMatrix) a database keyed on the Computer Vulnerability Enumeration (CVE) number, of various exploits used to gain access to systems; a suite of tools to assess the security of both binaries and source code; a property-based testing tool to slice software code looking for specific vulnerabilities using signatures from the VMatrix; an investigation into the verification of software designs for compliance to security properties, based on model checking approaches initially researched together with analytical verification of formal specification.

Current Sponsor:

NASA Office of Safety and Mission Assurance under the NASA Software Program lead by the NASA Software IV&V Facility.

Collaboration:

Collaborative support provided by the Computer Science Department, University of California Davis (UC Davis)

FY 2001 Center Software Initiative Proposal for the NASA Software IV&V Facility

Presentations and Papers:

- Analysis of Computer Vulnerabilities - Presentation by Matt Bishop
- Development of a Software Security Assessment Instrument to Reduce Software Security Risk - IEEE Paper

  Development of a Software Security Assessment Instrument to Reduce Software Security Risk - Presentation
- Reducing Software Security Risk Through an Integrated Approach - IEEE Paper

  Reducing Software Security Risk Through an Integrated Approach - Presentation
- NASA GSFC IV&V Facility Center Initiative Summary

Authors:

David P. Gilliam
Caltech, Jet Propulsion Laboratory
David.P.Gilliam@jpl.nasa.gov

John C. Kelly
Caltech, Jet Propulsion Laboratory
John.C.Kelly@jpl.nasa.gov

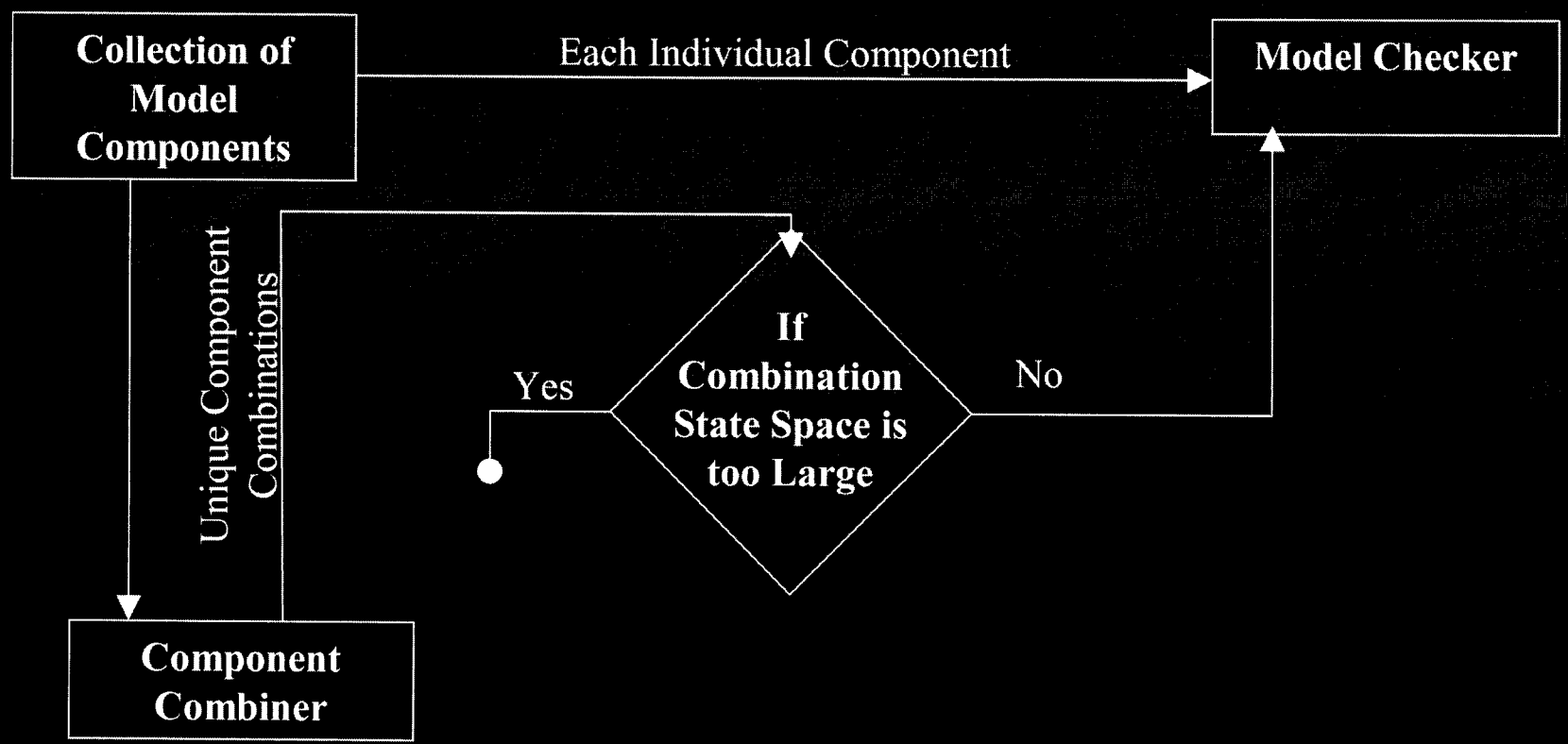
Matt Bishop
University of California at Davis
bishop@cs.ucdavis.edu

John D. Powell
Caltech, Jet Propulsion Laboratory
John.D.Powell@jpl.nasa.gov

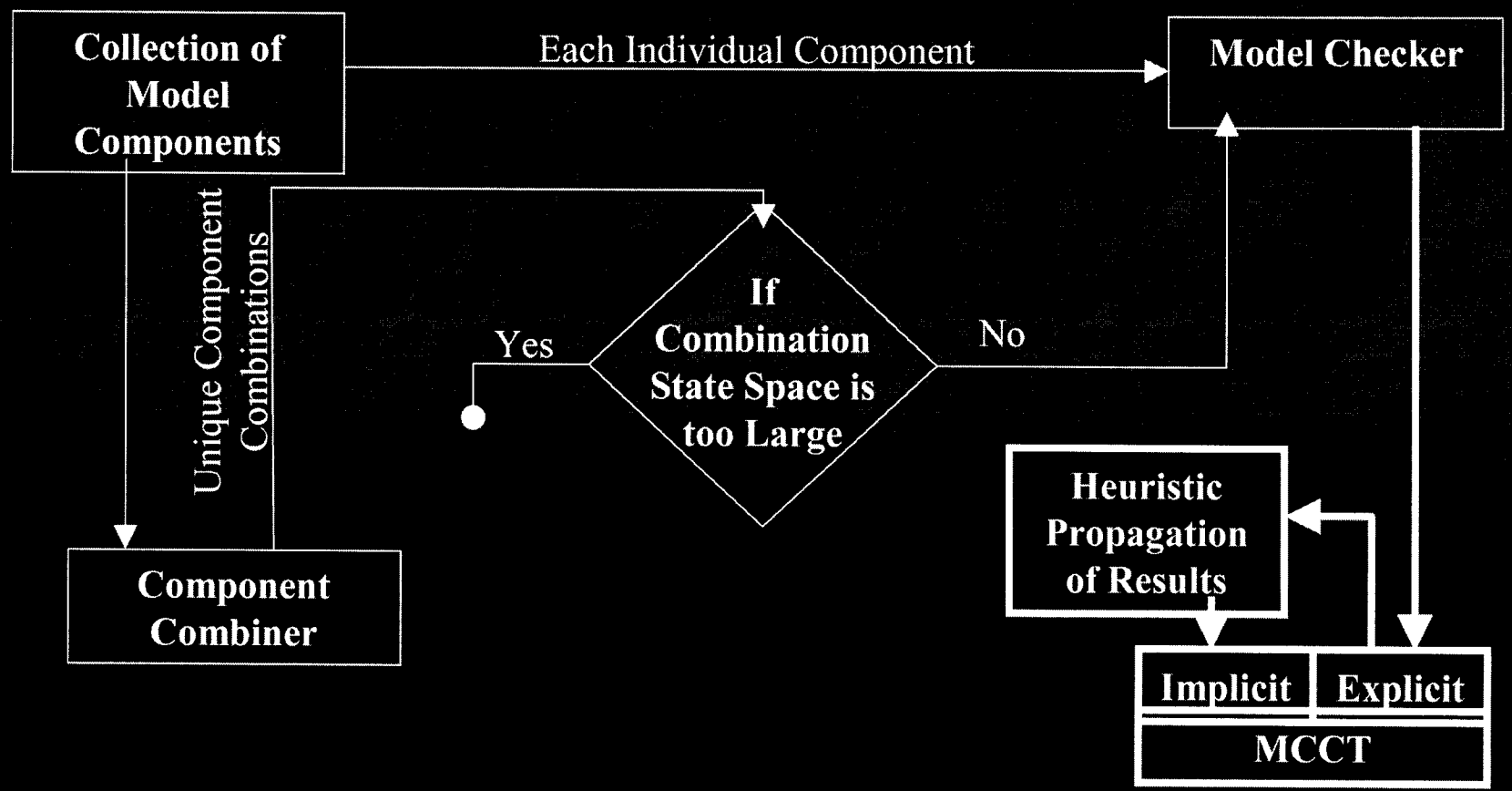
Deliverables:

- Vulnerability Matrix Report
- Paper "Reducing Software Security Risk Through an Integrated Approach" delivered to IEEE WET ICE international workshop on Enterprise Security
- Security Assessment Tools Report

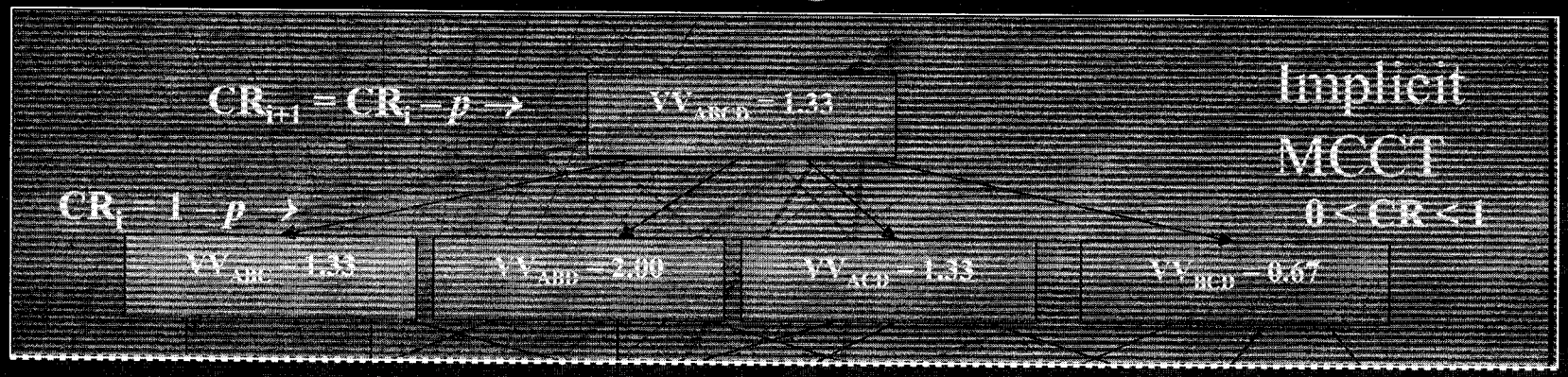
The Flexible Modeling Framework (FMF) – Part 1



The Flexible Modeling Framework (FMF) – Part 2



Heuristic VV and CR Propagation



Implicit MCCT

$0 < CR < 1$

$VV_{AB} = 2$ $VV_{AC} = 2$ $VV_{AD} = 2$ $VV_{BC} = 0$ $VV_{BD} = 2$ $VV_{CD} = 0$

$CR = 1 \rightarrow$

$CR = 1 \rightarrow$ $VV_A = 0$ $VV_B = 2$ $VV_C = 2$ $VV_D = 2$

Explicit MCCT

$CR = 1$

Heuristic Propagation

- Verification Values (VV) – degree to which a property holds or does not hold (0 – 2)
- Confidence Rating (CR) – degree of certainty that the VV is correct (0 – 1)
- Why 2 heuristic values instead of 1?
  - VV=1, CR=.99 – Confident inconclusiveness
    - Known information is highly contradictory
    - Variable model component resolution is unlikely to help
  - VV=1, CR=.01 – Inconclusive due to heuristics
    - Heuristic propagation's prediction accuracy is degrading
    - Variable model component resolution is more likely to help

Heuristic VV and CR Propagation

$CR_{i+1} = CR_i - p \rightarrow$ $VV_{ABCD} = 1.33$

Implicit MCCT

$0 < CR < 1$

$CR_i = 1 - p \rightarrow$

$VV_{ABC} = 1.33$ $VV_{ABD} = 2.00$ $VV_{ACD} = 1.33$ $VV_{BCD} = 0.67$

$VV_{AB} = 2$ $VV_{AC} = 2$ $VV_{AD} = 2$ $VV_{BC} = 0$ $VV_{BD} = 2$ $VV_{CD} = 0$

$CR = 1 \rightarrow$

$CR = 1 \rightarrow$ $VV_A = 0$ $VV_B = 2$ $VV_C = 2$ $VV_D = 2$

Explicit MCCT

$CR = 1$

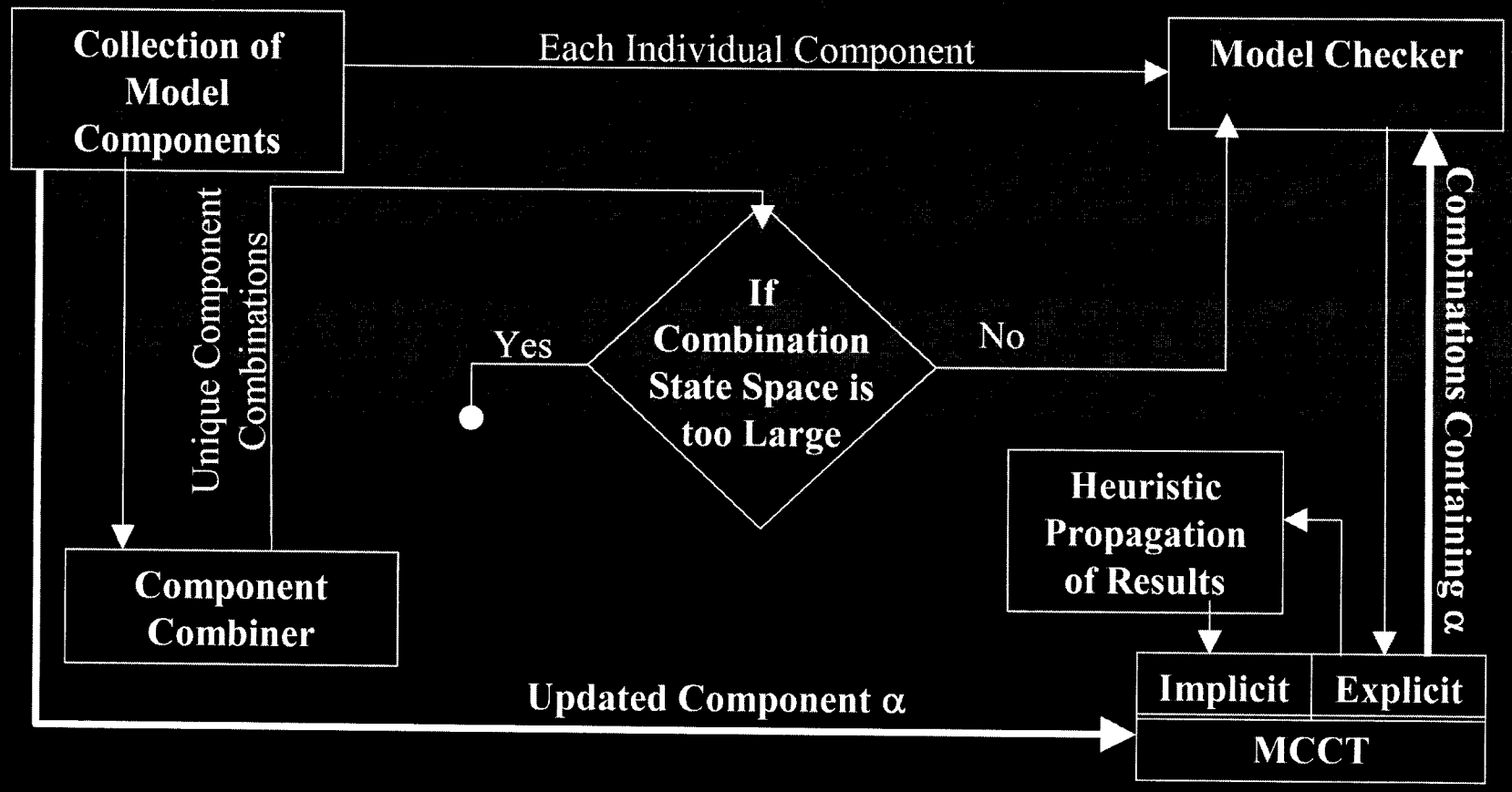
Variable Component Resolution

- Trading less abstract model components for more abstract ones.
  - Increased ability to analyze interaction of more model component
  - Decreased detail about the behavior of some model components
- Used to Cope with State Space Explosion

Do Model Component resolution versions occur as a side effect of model updates?

Rapid Model Updating

- FMF benefits software early in its lifecycle
  - Earlier Discovery of Software Errors
  - Correction is easier / better / less expensive
- FMF must adapt to early lifecycle events
  - Rapidly changing requirements and designs
  - Varying / Increasing levels of detail defined for different parts of the system.

How can traditional model development for MC be enhanced to cope with volatile systems?

The Flexible Modeling Framework (FMF) – Part 3