

# Deductive Frameworks for Vulnerability Analysis

C. R. Ramakrishnan

Department of Computer Science  
Stony Brook University



# Vulnerability Analysis

Vulnerabilities: potential security holes in computer system configurations.

- System functionality is provided by many components:  
e.g., user-level processes, OS kernel modules
- Security of the system may be compromised by
  - Software errors in individual processes.  
(e.g., buffer overflows, race conditions, trojans)
  - Improper system configuration.  
(e.g., file permission errors, contents of configuration files)
  - Integration errors arising from unexpected interactions among components.

## Vulnerability Analysis: The Problem

A large class of vulnerabilities arise due

- subtle interactions among multiple components: server processes, applications, filesystems, other OS services, etc.
  - improper system configuration
- Even if individual components operate correctly, system as a whole may exhibit undesirable behavior.
  - Complexity of interaction (concurrency, data sharing) makes manual debugging tedious, incomplete and impractical.
  - Formal analysis is the only way to systematically find such vulnerabilities.
  - *Can current analysis techniques be adapted for this problem?*
  - *What new techniques need to be developed?*

## Vulnerability Analysis: Our Approach

- Develop abstract models of individual system components.
  - Models may be vendor-provided, extracted from source code, or be developed based on our understanding of the component's functionality.
  - Models may be based on components'
    - \* *Operations and their relationships*
    - \* *Capabilities or information dependencies*
- Compose models together to derive global system behavior.
- Analyze system to check if the model compromises security objectives.

## Uncovering Interaction Vulnerabilities

- Obtain automata (process) models describing the behavior of components.

Such models can be obtained by

- Automatic static analysis of programs (including scripts)
  - Learning from program traces
  - Manual methods
- Global system behavior is composed from the models and represented as a transition relation.
    - Composition can be done on-the-fly.
  - Specify security properties as high-level policy statements.  
Example:
    - If a system-critical file is overwritten, the data must come from a protected source.
  - Check if the stated policies hold using *model checking*.

## Detecting Vulnerability Propagation

Vulnerabilities in certain components of a system (e.g. network services) can be used to launch multi-step attacks.

Basic Data used by the analysis:

- Collection of services, and their attributes
- Network configuration
- Known vulnerabilities in programs (CVE)
- Users and their privileges

Propagation of vulnerabilities can be formulated as *rules*.

Queries to determine whether users have more capabilities than their given privileges will reveal the effects of vulnerabilities.

## Vulnerability Propagation Rules

```
fullControl(Principal, Host, Priv) :-  
    vulExists(Host, VulID, Program),  
    vulProperty(VulID, remoteExploit, privEscalation),  
    clientProgram(Host, Program, Priv).
```

```
fullControl(Principal, Host, Priv) :-  
    fullControl(Principal, Host, SomePriv),  
    vulExists(Host, VulID, Program),  
    vulProperty(VulID, localExploit, privEscalation),  
    clientProgram(Host, Program, Priv).
```

- ? How do we arrive at these rules?
- ? What is the effect of modifying these rules or the base configuration data?
- ? How do we isolate the vulnerable systems with minimal changes to the system's usability?

## Deductive Spreadsheets (DSS)

- A spreadsheet-like paradigm for programming with **rules**.
- Similar to a spreadsheet, the user can write and modify rules based on *instances* and immediately see their *effects*.
- A small number of significant, yet natural, extensions to the spreadsheet metaphor:
  - A cell in a spreadsheet may contain a **set** of values.
  - When one cell **A** refers to another cell **B**, it means that **A** **contains** all elements of **B**.
  - Cell references may be recursive (meaning: least fixed point).
  - Rows, columns, sheets etc are *named*. Their names can appear as values in cells.
  - Elements in a set may have structured values (i.e. tuples).

# Example: Network Configuration

	A	B	C	D
1		webserver	fileserver	workstation
2	webserver	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[(rpc, 100003), (rpc, 100005)]	
3	fileserver	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[(rpc, 100003), (rpc, 100005), (tcp, 80)]
4	workstation	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[(rpc, 100003), (rpc, 100005), (tcp, 80)]
5	internet	(tcp, 80)		

Navigation: \had \vuln \netsvc\_port \netsvc\_priv \uaccounts \\_exec

	A	B	C	D	E
1		httpd	nfsd	mountd	
2	webserver	(tcp, 80)			
3	fileserver		(rpc, 100003)	(rpc, 100005)	
4	workstation				
5	internet				

Navigation: \had \vuln \netsvc\_port \netsvc\_priv \uaccounts \\_exec

	A	B	C	D	E
1		remote_priv	local_priv		
2	webserver	httpd			
3	fileserver	mountd			
4	workstation				
5	internet				

Navigation: \had \vuln \netsvc\_port \netsvc\_priv \uaccounts \\_exec

	A	B	C	D	E
1		webserver	fileserver	workstation	internet
2	normaluser			userAccount	
3	sysadmin	root	root	root	
4	attacker				root
5					

Navigation: \had \vuln \netsvc\_port \netsvc\_priv \uaccounts \\_exec

# Example: Vulnerability Analysis

	A	B	C	D	E	F	G	H
1		exec1	accessible_ports	accessible_pgms	vuln_pgms	exec2	exec_all	exec_hosts_priv
2	webserver	[]	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	httpd	httpd	apache	apache	(webserver, apache)
3	fileserver	[]	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[mountd, nfsd]	mountd	root	root	(fileserver, root)
4	workstation	userAccount	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[]	[]	[]	userAccount	(workstation, userAccount)
5	internet	[]	[(rpc, 100003), (rpc, 100005), (tcp, 80)]	[]	[]	[]	[]	[]

  

	A	B	C
1		hosts_privs	hosts
2	normaluser	[(fileserver, root), (webserver, apache), (workstation, userAccount)]	[fileserver, webserver, workstation]
3	sysadmin	[(fileserver, root), (webserver, apache), (webserver, root), (workstation, root)]	[fileserver, webserver, workstation]
4	attacker	[(fileserver, root), (internet, root), (webserver, apache)]	[fileserver, internet, webserver]

  

	A	B	C	D	E
1		webserver	fileserver	workstation	internet
2	normaluser	apache	root	userAccount	[]
3	sysadmin	[apache, root]	root	root	[]
4	attacker	apache	root	[]	root

1. uaccounts!(webserver normaluser)
2. hacl!(webserver (uaccess!(hosts normaluser)))
3. netsvc\_port!((accessible\_ports webserver) ~webserver)
4. (accessible\_pgms webserver) & (vuln!(remote\_priv webserver))
5. netsvc\_priv!((vuln\_pgms webserver) webserver)

## Proposed Project

A DSS-based system for vulnerability analysis.

- View and manage *intentions* (rules) as well as *extensions* (their effects).
- Extract configuration data into spreadsheets.
- Extract explanations for access violations to isolate vulnerabilities.
- Improve trustworthiness of the rules themselves (e.g. using types).

Other Applications: Results can also be used to construct a system for security policy development and analysis (e.g. in trust management).