14th USENIX Security Symposium, August 2005

MulVAL: A logic-based network security analyzer

Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel
Princeton University

Outline

- Introduction
- Representation
- Vulnerability Specification
- The MulVAL Reasoning System
- Examples
- Hypothetical Analysis
- Performance and Scalability
- Related Work
- Conclusion

Introduction

- Two features are critical for vulnerability analysis tool
  - Can automatically integrate formal vulnerability spec
  - Be able to scale to networks with thousands of machine
- MulVAL
  - An end-to-end framework and reasoning system
    - Conducts multi-host, multi-stage vulnerability analysis
  - Adopt Datalog as modeling language
    - Bug spec, configuration, reasoning rules, system permission, privilege
  - The authors can leverage existing vulnerability database and scanning tools by Datalog and feeding it into MulVAL reasoning engine to perform analysis in seconds.
    - for networks with thousands of machines

Introduction

- One of a sysadmin's daily chores is
  - to read bug reports from various sources
    - such as CERT, BugTraq etc
  - to understand which reported bugs are actually security vulnerabilities in the context of his own network
  - to assessment of their security impact on the network
  - patch and reboot, reconfigure a firewall, dismount a file-server partition, and so on

- A vulnerability analysis tool can be useful,
  - if it can automatically do so,
  - and only if it is scalable.

The inputs to MulVAL's analysis are

- Advisories
  - What vulnerabilities have been reported and do they exist on my machines?
- Host configuration
  - What software and services are running on my hosts, and how are they configured?
- Network configuration
  - How are my network routers and firewalls configured?
- Principals
  - Who are the users of my network?
- Interaction
  - What is the model of how all these components interact?
- Policy
  - What accesses do I want to permit?

Representation (1/2)

- Advisories
  - vulExists(webServer, 'CAN-2002-0392', httpd)
  - vulProperty('CAN-2002-0392', remoteExploit, privilegeEscalation)
- Host configuration
  - networkService(webServer, httpd, TCP, 80, apache)
- Network configuration
  - hacl(internet, webServer, TCP, 80) // host access control lists
- Principals
  - hasAccount(user, projectPC, userAccount)
  - hasAccount(sysAdmin, webServer, root)

Representation (2/2)

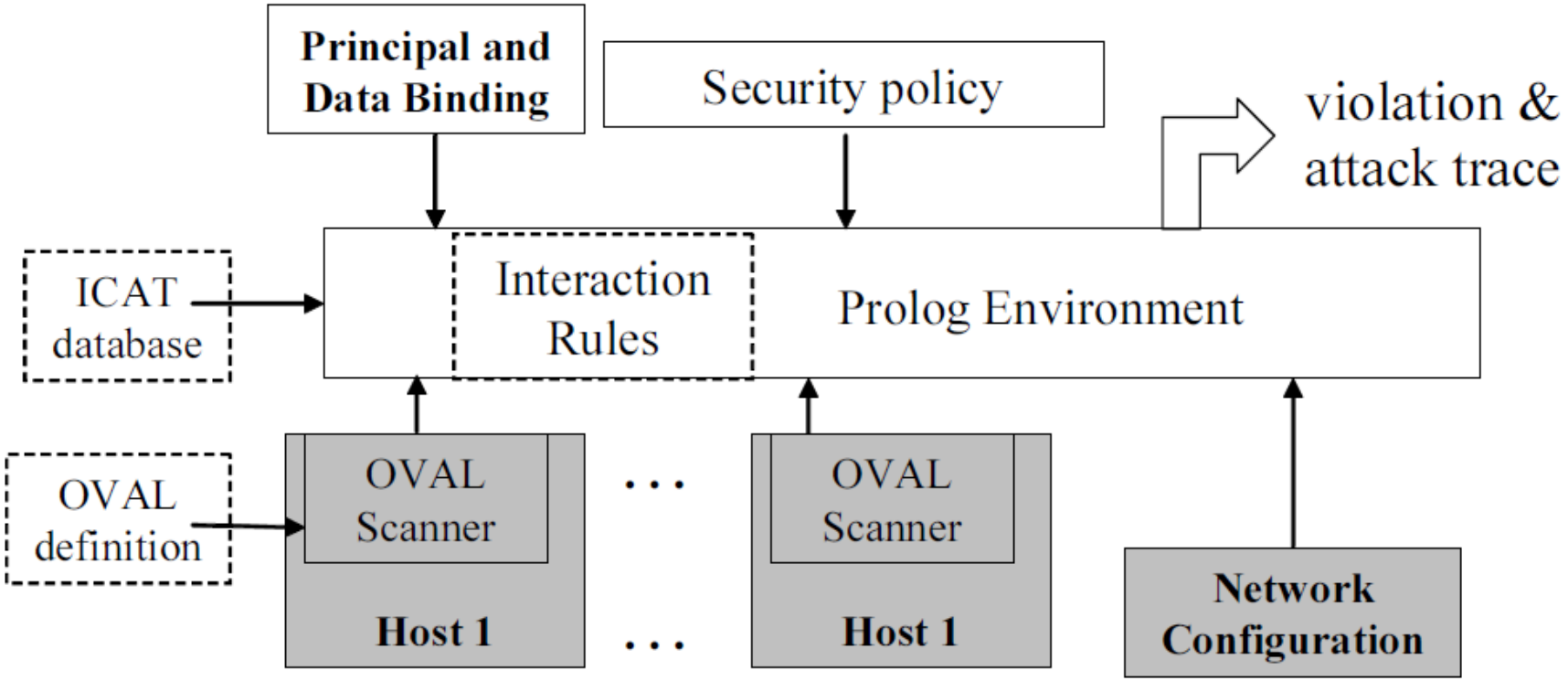
- Interaction
  - execCode(Attacker, Host, Priv) :-
    vulExists(Host, VulID, Program),
    vulProperty(VulID, remoteExploit, privEscalation),
    networkService(Host, Program, Protocol, Port, Priv),
    netAccess(Attacker, Host, Protocol, Port),
    malicious(Attacker).
- Policy
  - allow(Everyone, read, webPages)
  - allow(systemAdmin, write, webPages)

Vulnerability Specification

- A specification of a security bug consists of two parts
  - how to recognize the existence of the bug on a system
  - what is the effect of the bug on a system

- Formal, machine-readable formats
  - OVAL (Open Vulnerability Assessment Language)
    - a formal specification language for recognizing vulnerabilities
    - http://oval.mitre.org/documents/docs-03/intro/intro.html
  - ICAT (or National Vulnerability Database)
    - a database that provides a vulnerability's effect
    - http://icat.nist.gov/icat.cfm

The MulVAL framework



The OVAL language and scanner

- XML-based language
- an OVAL definition can specify how to check a machine for the existence of a new software vulnerability
- an OVAL-compatible scanner will conduct the specified tests and report the result

  - networkService(Host, Program, Protocol, Port, Priv).
  - clientProgram(Host, Program, Priv).
  - setuidProgram(Host, Program, Owner).
  - filePath(H, Owner, Path).
  - nfsExport(Server, Path, Access, Client).
  - nfsMountTable(Client, ClientPath, Server, ServerPath).

Vulnerability effect (in ICAT)

- exploitable range
  - Local: a local exploit requires that the attacker already have some local access on the host
  - Remote
- consequence
  - confidentiality loss
  - integrity loss
  - denial of service
  - privilege escalation

Example:

```
vulProperty('CVE-2004-00495',
                localExploit,
                privEscalation).
```

The MulVAL Reasoning System

- A *literal*, $p(t_1, \ldots, t_k)$ is a predicate applied to its arguments, each of which is either a constant or a variable.

- Let $L_0, \ldots, L_n$ be literals, a sentence in MulVAL is represented as L_0 :- $L_1, \ldots, L_n$
  - Semantically, it means if $L_1, \ldots, L_n$ are true then L_0 is also true.

  - A clause with an empty body (right-hand side) is called a *fact*.
  - A clause with a nonempty body is called a *rule*.

Exploit rules

- execCode(P, H, UserPriv)
  - Principal P can execute arbitrary code with privilege UserPriv on machine H
- netAccess(P, H, Protocol, Port)
  - Principal P can send packets to Port on machine H through Protocol

Example: remote exploit of a client program

```
execCode(Attacker, Host, Priv) :-
        vulExists(Host, VulID, Program),
        vulProperty(VulID, remoteExploit, privEscalation),
        clientProgram(Host, Program, Priv),
        malicious(Attacker).
```

* 84% of vulnerabilities are labeled with privilege escalation or only labeled with DoS

Multistage attacks

- if an attacker P can access machine H with Owner's privilege, then he can have arbitrary access to files owned by Owner.
  - accessFile(P, H, Access, Path) :-
    execCode(P, H, Owner),
    filePath(H, Owner, Path).

- if an attacker can modify files under Owner's directory, he can gain privilege of Owner.
  - execCode(Attacker, H, Owner) :-
    accessFile(Attacker, H, write, Path),
    filePath(H, Owner, Path),
    malicious(Attacker).

Host Access Control List/ Policy spec

- hacl(Source, Destination, Protocol, DestPort)

- Multihop network access
  - netAccess(P, H2, Protocol, Port) :-
    execCode(P, H1, Priv),
    hacl(H1, H2, Protocol, Port).

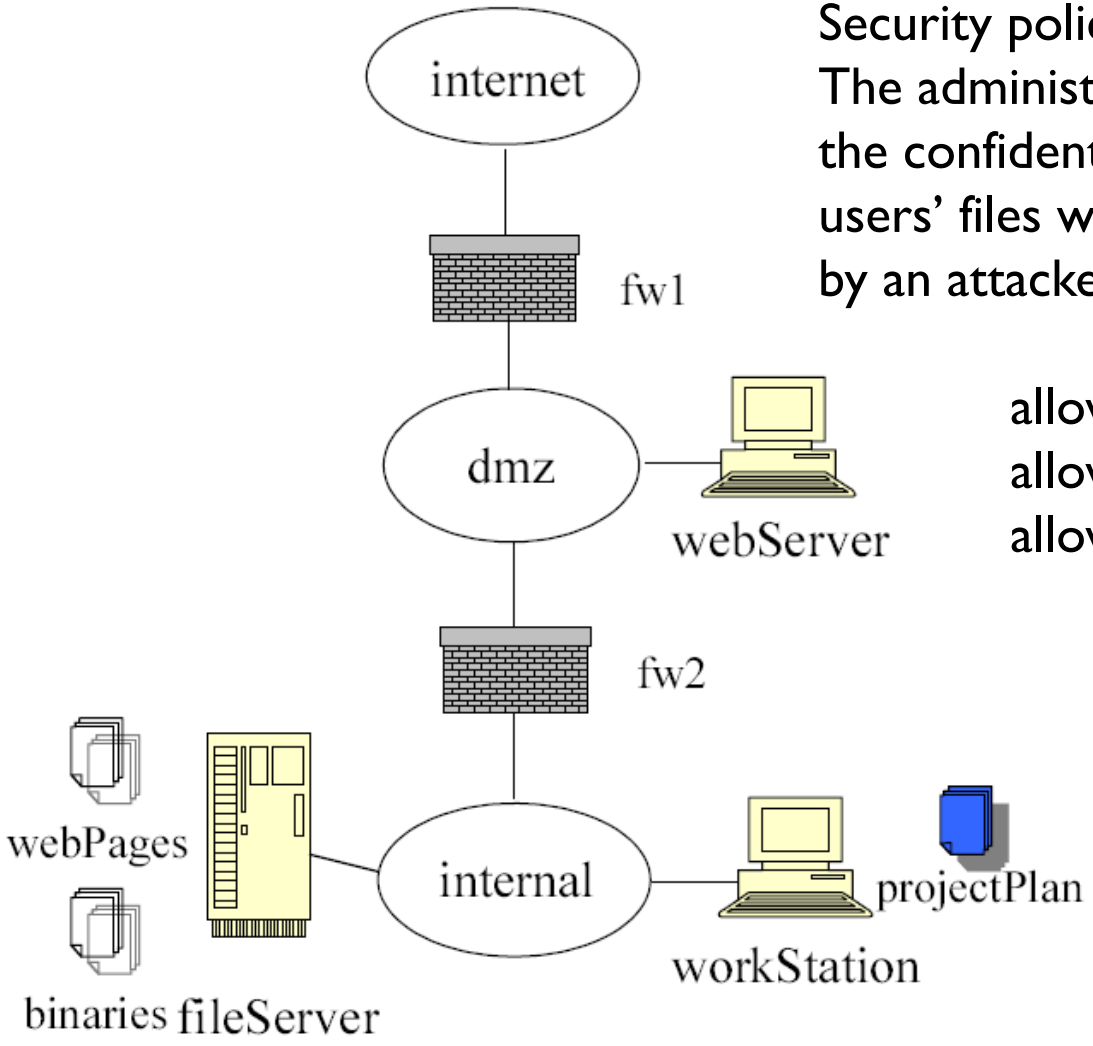
- allow(Principal, Access, Data)
  - allow(Everyone, read, webPages).
  - allow(user, Access, projectPlan).
  - allow(sysAdmin, Access, Data).

Binding information / Algorithm

- hasAccount(user, projectPC, userAccount).
- hasAccount(sysAdmin, webServer, root).
- dataBind(projectPlan,workstation,'/home').
- dataBind(webPages, webServer, '/www').

- access(P, Access, Data) :-
  dataBind(Data, H, Path),
  accessFile(P, H, Access, Path).
- policyViolation(P, Access, Data) :-
  access(P, Access, Data),
  not allow(P, Access, Data).

Example



Security policy:
The administrators need to ensure that the confidentiality and the integrity of users' files will not be compromised by an attacker.

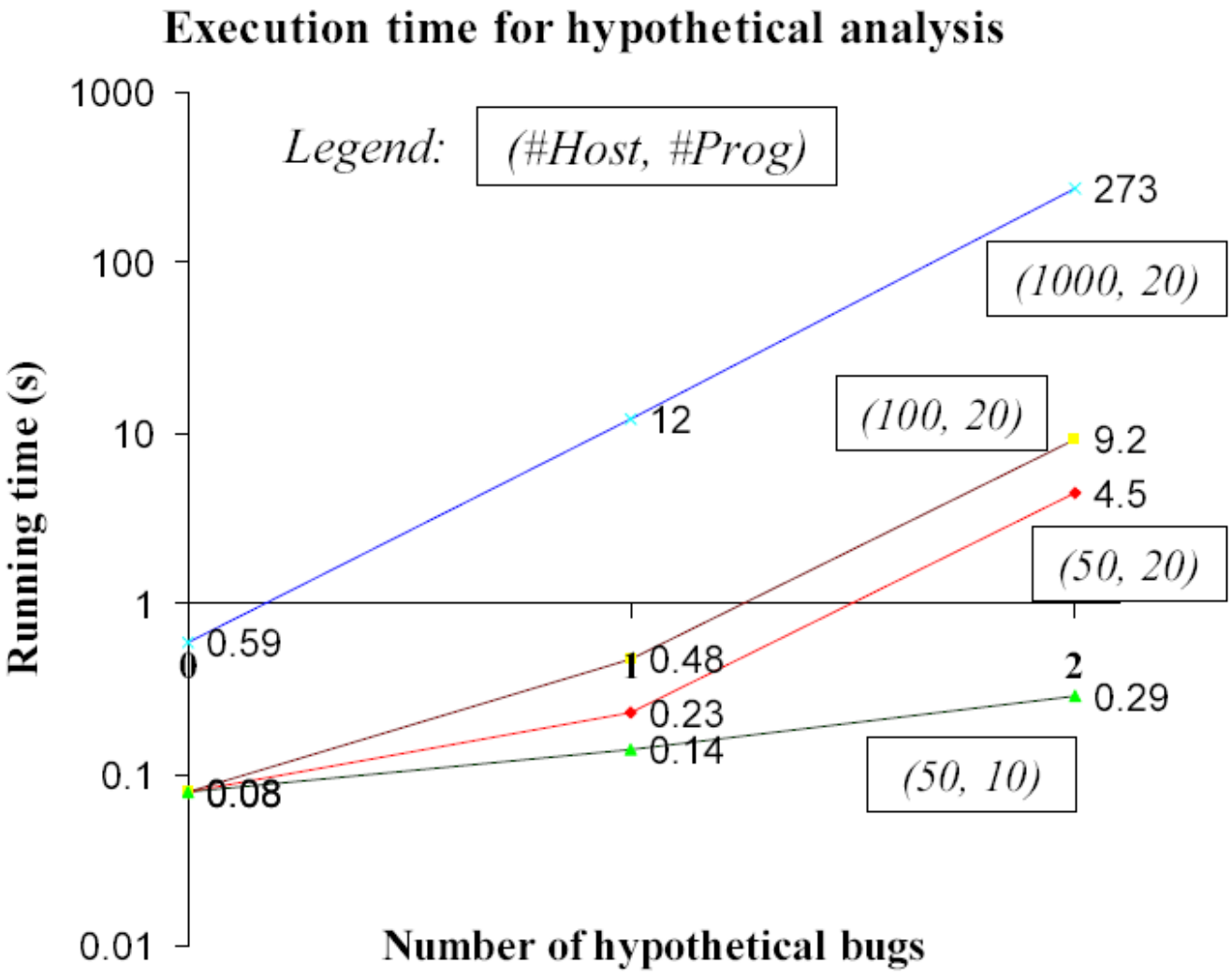
allow(Anyone, read, webPages).
allow(user, AnyAccess, projectPlan).
allow(sysAdmin, AnyAccess, Data).

MulVAL scanner		236 s
MulVAL reasoning engine	§5.1	0.08
	1 host	0.08
	200 hosts	0.22
	400 hosts	0.75
	1000 hosts	3.85
	2000 hosts	15.8

Hypothetical analysis

- One important usage of vulnerability reasoning tools is to conduct "what if" analysis.
  - The authors introduce a predicate *bugHyp* to represent hypothetical software vulnerabilities
  - vulExists(Host, VulID, Prog) :-
    bugHyp(Host, Prog, Range, Consequence).
  - vulProperty(VulID, Range, Consequence) :-
    bugHyp(Host, Prog, Range, Consequence).

Execution time for hypothetical analysis



Since the hypothetical analysis goes through all combination of programs to inject bugs, the running time is dependent on both the number of programs and the number of hypothetical bugs.

Related Work

- Old works did not how to automatically integrate vulnerability specifications from the bug-reporting community into the reasoning model.

- The difference between Datalog and model-checking is that derivation in Datalog is a process of accumulating true facts.
  - Since the number of facts is polynomial in the size of the network, the process will terminate efficiently.
- Model checking checks temporal properties of every possible state-change sequence.
  - The number of all possible states is exponential in the size of the network

Related Work (cont'd)

- For network attacks, one can assume the monotonicity property—gaining privileges does not hurt an attacker's ability to launch more attacks.
- If at a certain stage an attacker has multiple choices for his next step, the order in which he carries out the next attack steps is irrelevant for vulnerability analysis under the monotonicity assumption.
- While it is possible that a model checker can be tuned to utilize the monotonicity property and prune attack paths that do not need to be examined
  - model checking is intended to check rich temporal properties of a state-transition system.

Conclusion

- We have demonstrated how to model a network system in Datalog so that network vulnerability analysis can be performed automatically and efficiently.

- A simple Prolog programcan perform "what-if" analysis for hypothetical software bugs efficiently.

Comments

- Including all the possible "elements" to describe attack's behaviors/host configurations/vulnerability/network.
  - It's difficult to design a model to fit into all different kinds of attacks.
- The "security policy" is a little bit weak (only base on access privilege)?
  - Limit to vulnerability-exploit attack
- Using Prolog is a good design.
- "What if" is attractive.