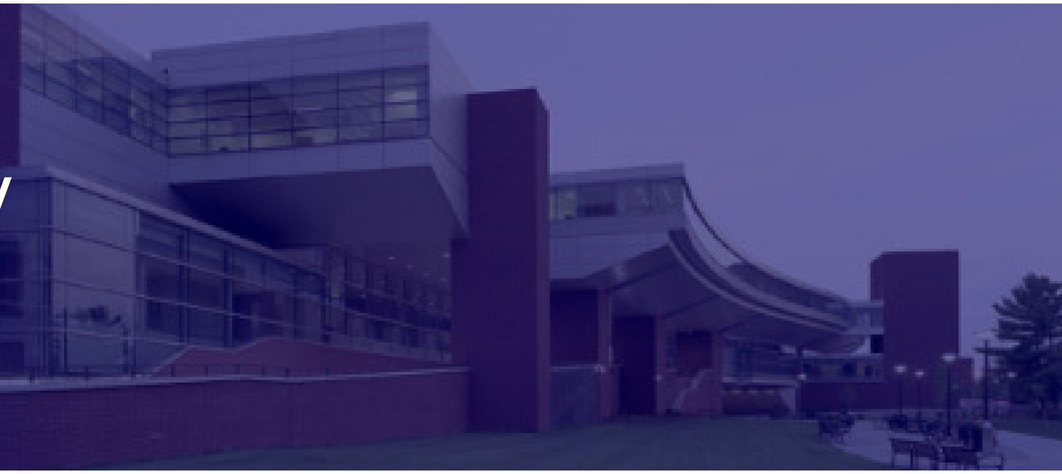
Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

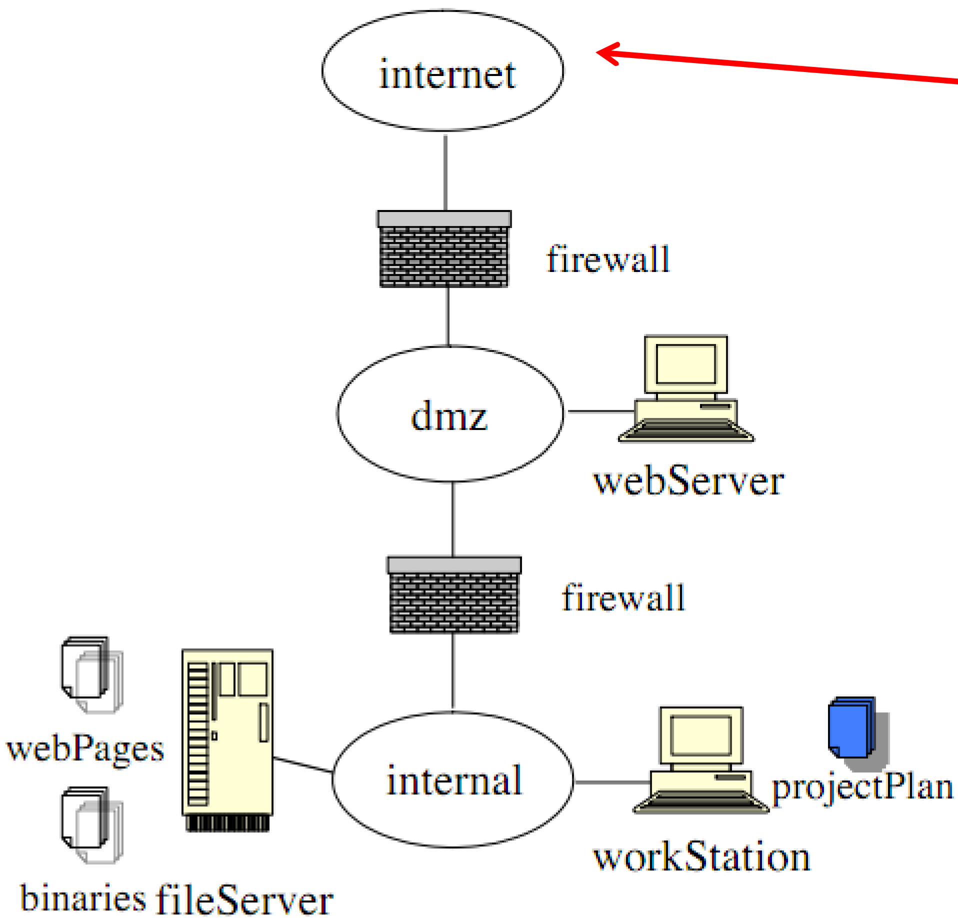


A Scalable Approach to Attack Graph Generation

By Ou, Boyer, McQueen

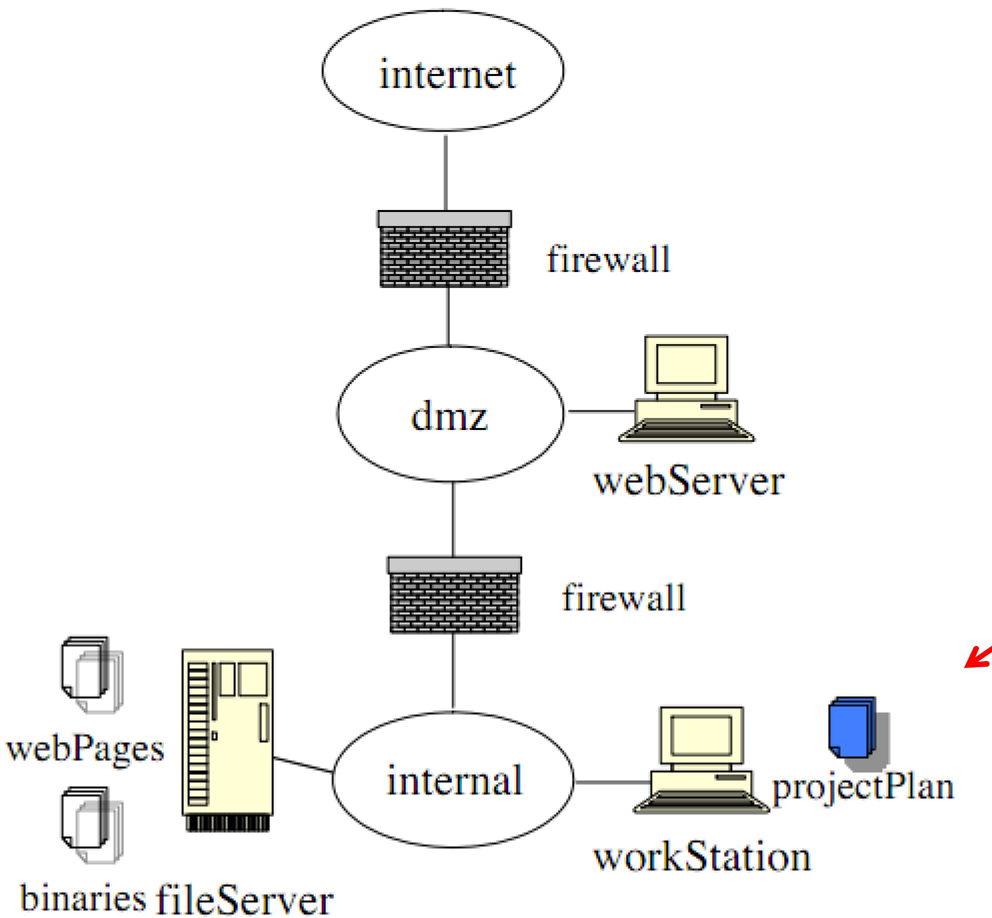
Presented By: Philip Koshy

(De)motivating Example



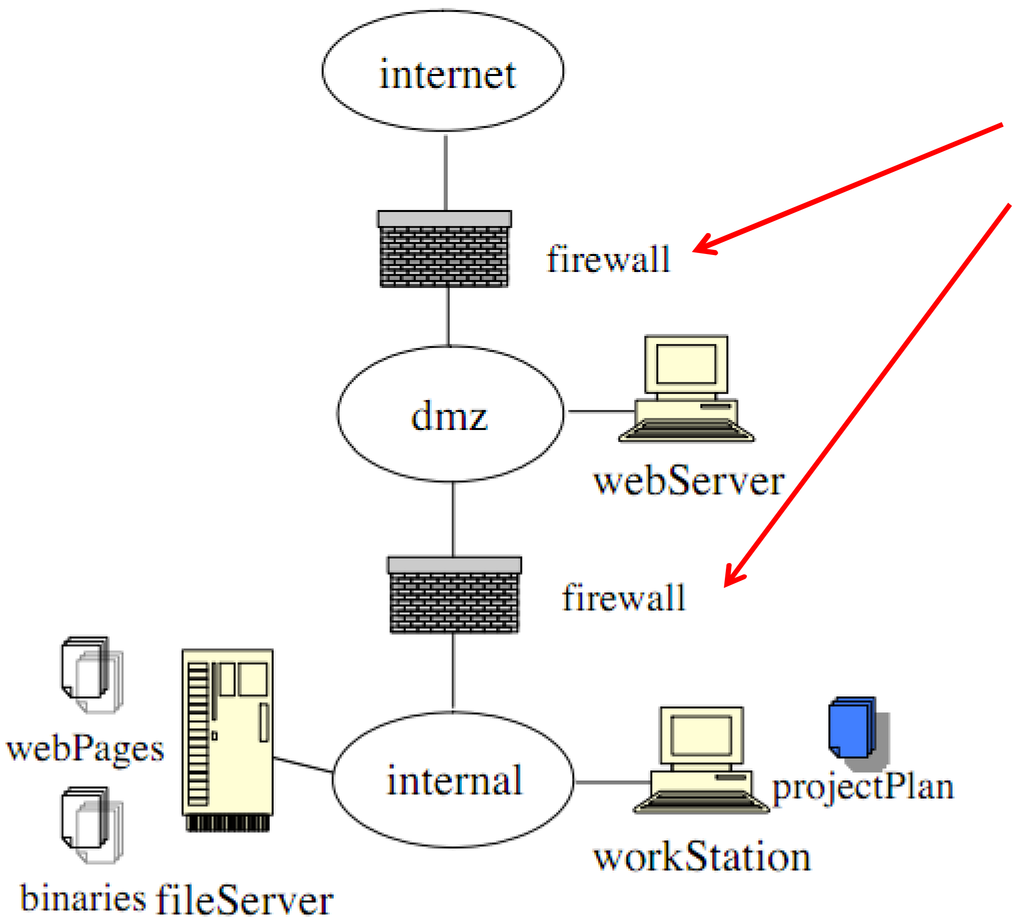
An attacker exists somewhere on the internet.

(De)motivating Example



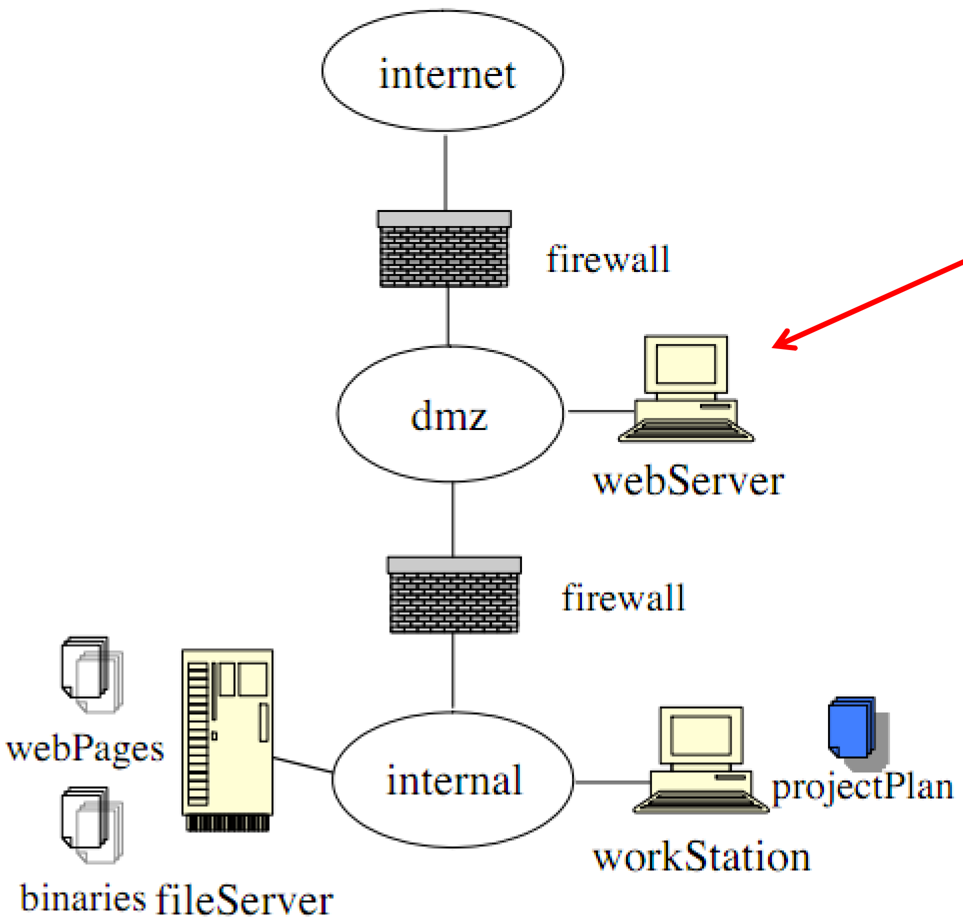
The attacker wants access to the Project Plan.

(De)motivating Example



Two firewalls in his/her way.

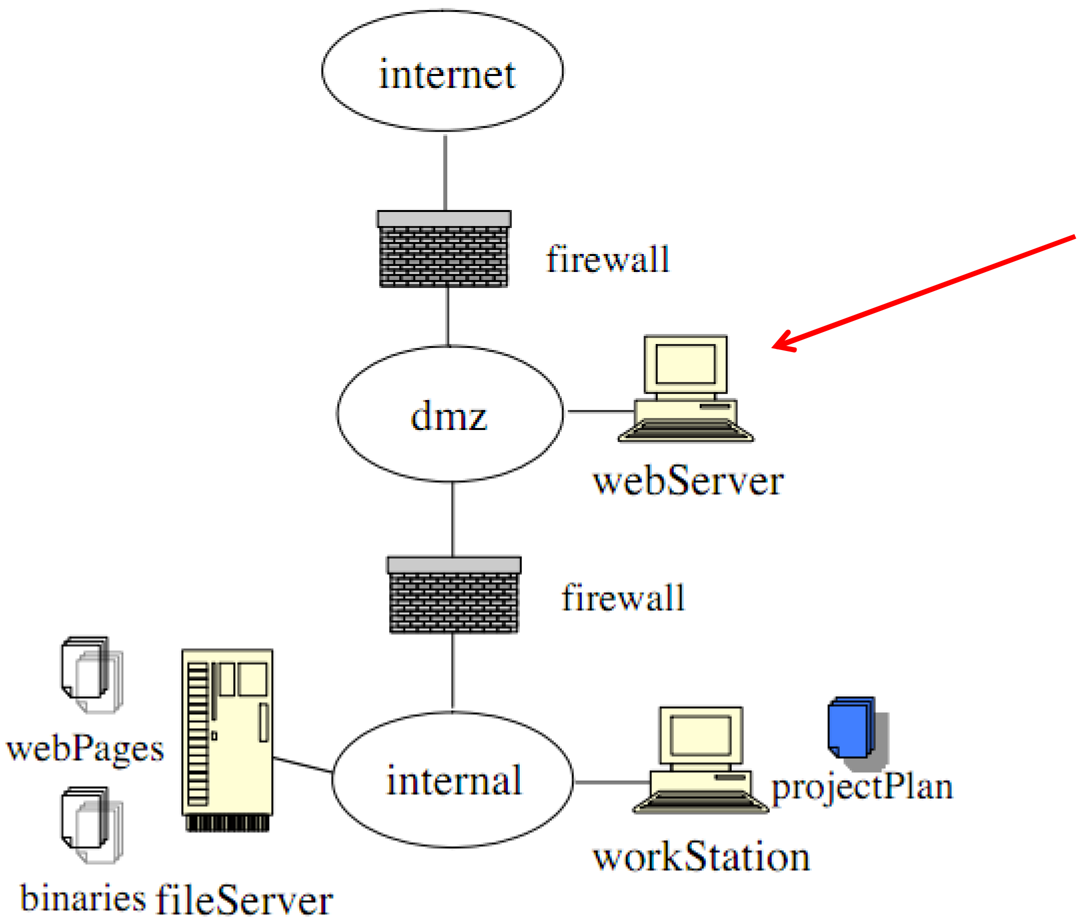
(De)motivating Example



The web server is the only server that is publicly accessible.

This is the first target.

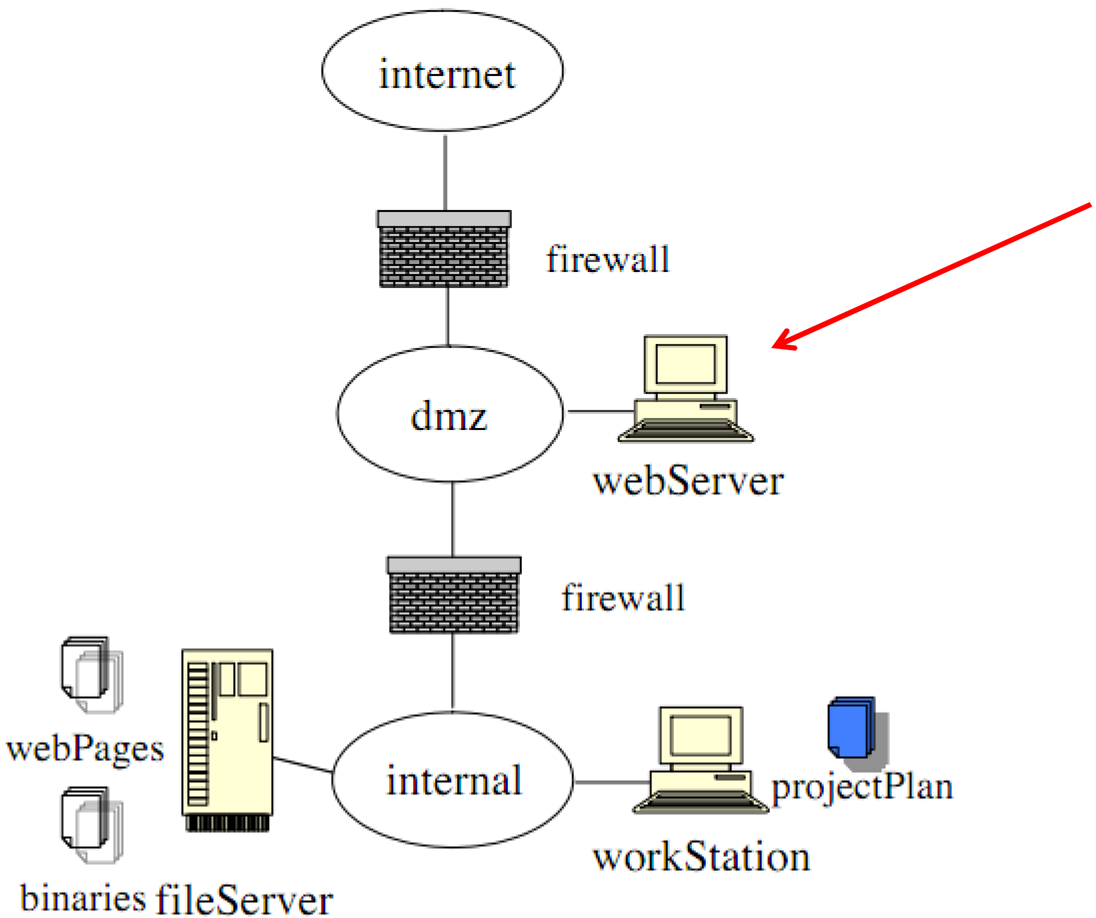
(De)motivating Example



The attacker successfully executes a remote exploit on the web server.

He/she now has local access to the web server.

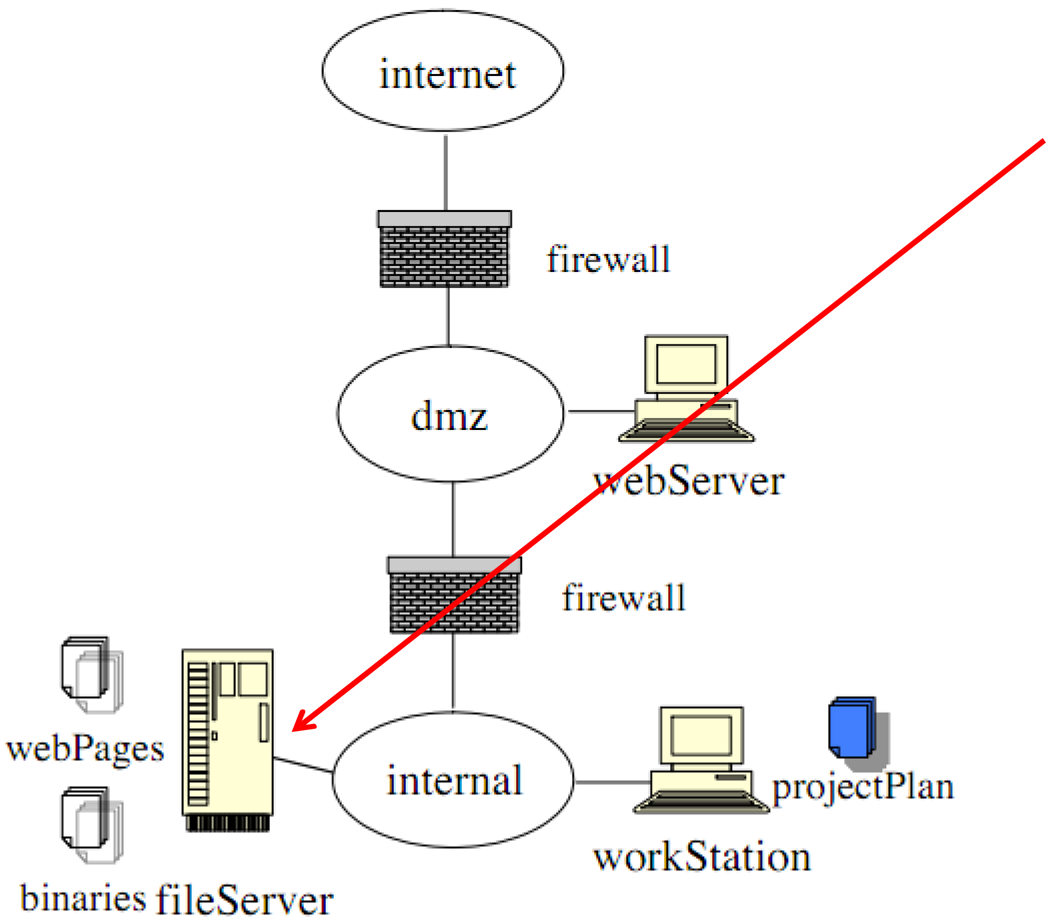
(De)motivating Example



After gaining access, the attacker notices that the web server can communicate with the file server using NFS.

They have just identified their next target!

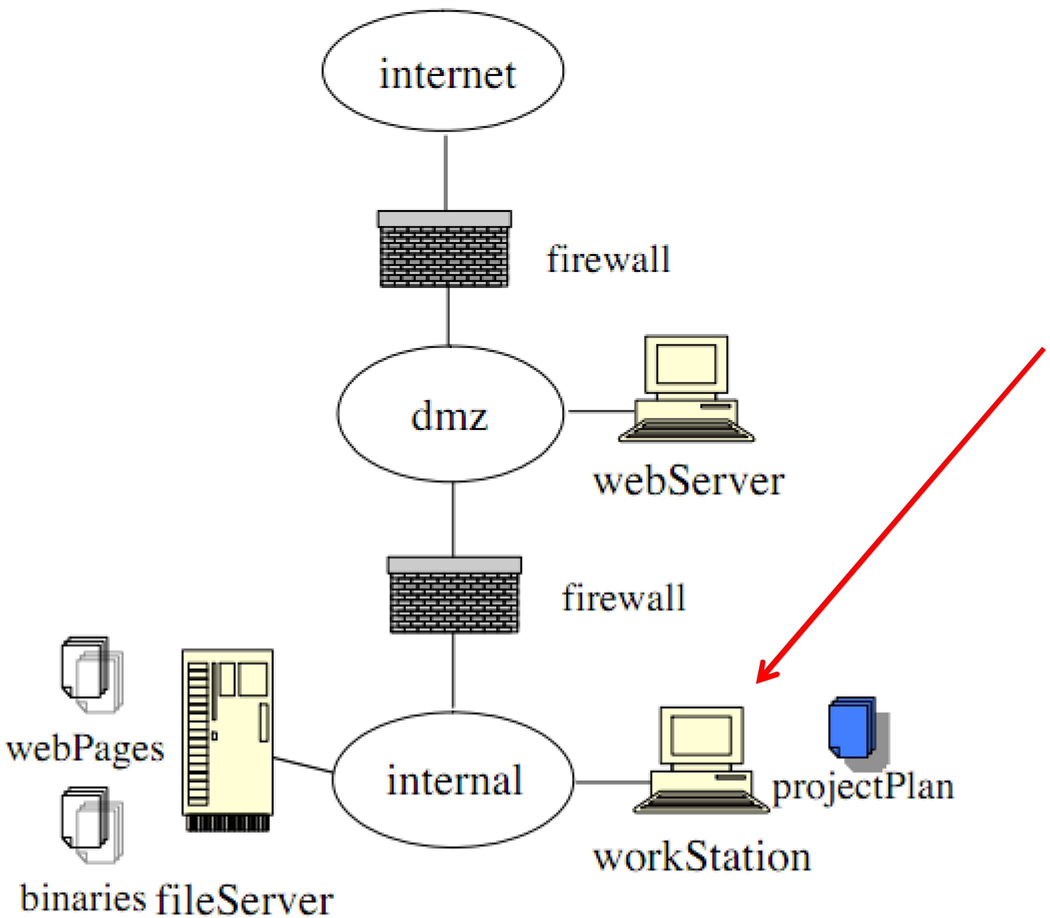
(De)motivating Example



The attacker notices that NFS on the file server is misconfigured.

The attacker places a modified binary (trojan horse) on the file server.

(De)motivating Example



The unsuspecting user on the workstation runs the trojan horse, which secretly exfiltrates the project plan to the attacker.

Things to note

- The attack was multi-stage.
  - The attack had a distinct procedure that moved in ordered stages.
- The attack was multi-host.
  - The attacker broke into/circumvented several systems.
- This is becoming more common and more dangerous (e.g., Stuxnet)

Main Idea

- Configuration errors cause security issues
- Attackers take the path of least resistance to reach their goal
- The security of an entire network may boil down to configuration errors on a single node (i.e., the weakest link)

Main Idea

PENNSTATE 1855

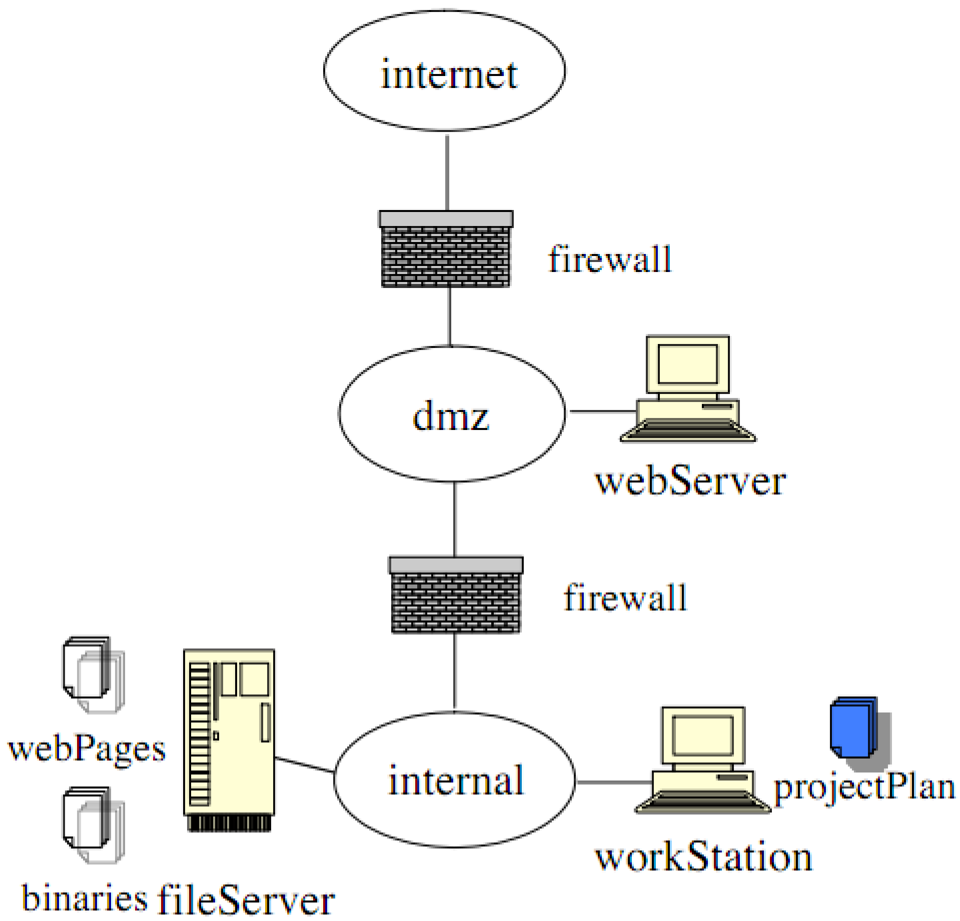
- The complexity of manually defending against configuration errors is non-trivial.
- Automated tools are necessary.
- The goal would be to answer two questions:
  - Is our network vulnerable to currently known attacks?
  - If so, how? We should have a clearly identified "path."

Main Idea

- The paper briefly discusses existing tools and indicates their limitations.
  - They often have incomprehensible output
  - Require non-standardized, ad-hoc inputs
  - No formal foundation

- Most important issues is scalability.
  - Existing tools could not handle networks with more than 20 nodes!

Main Idea



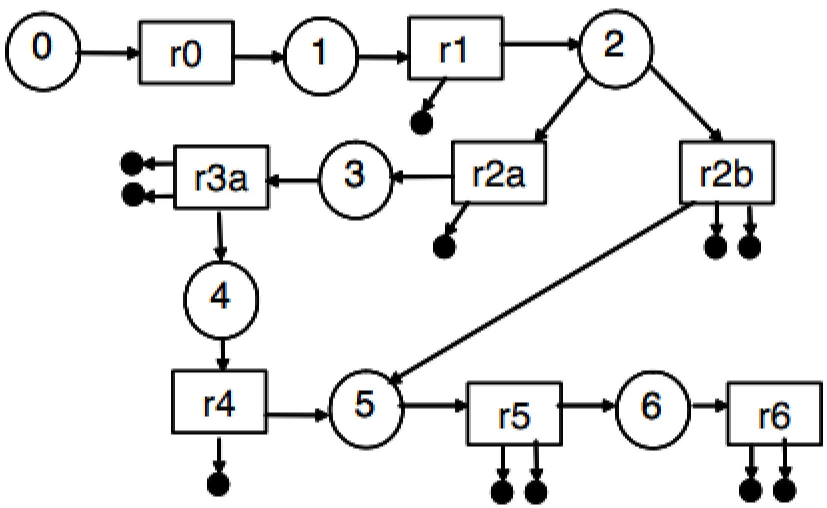


Figure 4: An example logical attack graph

Human readable output

```
<0>|--execCode(attacker,workStation,root)
   <r0>Rule5: Trojan horse installation
     <1>|--accessFile(attacker,workStation,write,/usr/local/share)
        <r1>Rule14: NFS semantics
             []-nfsMounted(workStation,/usr/local/share,fileServer,/export,read)
          <2>||--accessFile(attacker,fileServer,write,/export)
             <r2a>Rule10: execCode implies file access
                  []-fileSystemACL(fileServer,root,write,/export)
               <3>|--execCode(attacker,fileServer,root)
                  <r3>Rule3: remote exploit of a server program
                       []-networkServiceInfo(fileServer,mountd,rpc,100005,root)
                       []-vulExists(fileServer,CVE-2003-0252,mountd,
                                    remoteExploit,privEscalation)
                    <4>|--netAccess(attacker,fileServer,rpc,100005)
                       <r4>Rule6: multi-hop access
                            []-hacl(webServer,fileServer,rpc,100005)
                         <5>|--execCode(attacker,webServer,apache)
                            <r5>Rule3: remote exploit of a server program
                                 []-networkServiceInfo(webServer,httpd,tcp,80,apache)
                                 []-vulExists(webServer,CAN-2002-0392,httpd,
                                                      remoteExploit,privEscalation)
                              <6>|--netAccess(attacker,webServer,tcp,80)
                                 <r6>Rule7: direct network access
                                      []-hacl(internet,webServer,tcp,80)
                                      []-located(attacker,internet)
```

Closest competitor

- The closest competitor (Sheyner et al.) has a formal foundation, but is impractical.
- Using Sheyner's approach, a network of only 10 hosts with 5 vulnerabilities per host took 15 minutes to analyze and generated 10 million edges.
- The major problem: Many duplicate paths of the graph are traversed!
  - Solution: Memoization!

How to proceed?

PENNSTATE 1855

- To answer these questions
  - We need to examine our configuration data
  - Define current vulnerabilities
  - Derive all potential attack graph through our network by combining our configurations with vulnerabilities.

Architecture

Security advisories

General information about recent vulnerabilities

Network configuration

Machine configuration

Specific information about your network configuration.

Side note

- General information about vulnerabilities is available in a computer digestible format (XML) through the MITRE corporation.

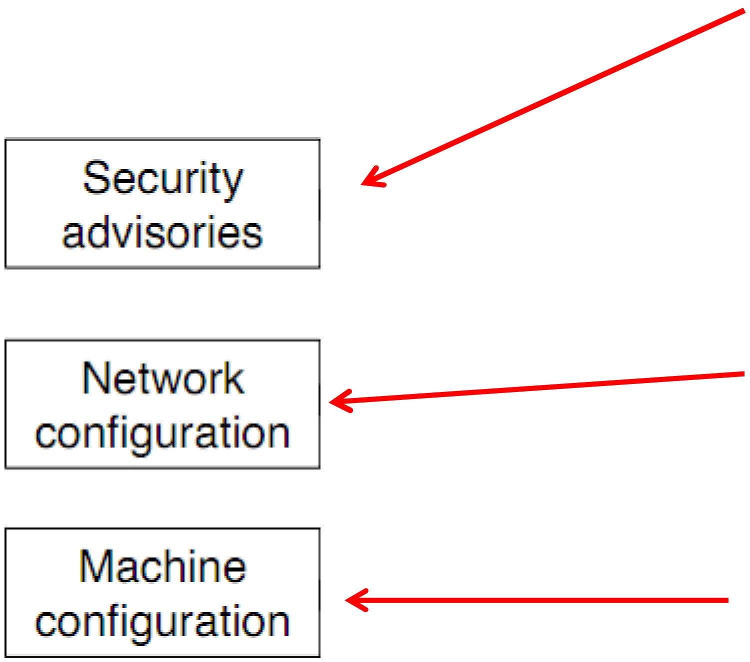
- Example vulnerability description:

  oval:org.mitre.oval:def:12860
  “Heap-based buffer overflow in the Web Audio implementation in Google Chrome before 15.0.874.102”

Side note


```
▼<registry_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" id="oval:org.mitre.oval:obj:15822" version="1" comment="The registry key to check if Google Chrome is installed (admin install for all users)">
  <hive>HKEY_LOCAL_MACHINE</hive>
 ▼<key>
    SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\Google Chrome
  </key>
  <name>DisplayName</name>
</registry_object>
▼<registry_object xmlns="http://oval.mitre.org/XMLSchema/oval-definitions-5#windows" id="oval:org.mitre.oval:obj:15382" version="2" comment="The registry key to check if Google Chrome is installed (individual users install)">
  <hive>HKEY_USERS</hive>
 ▼<key operation="pattern match">
    ^S-.*\\Software\\Microsoft\\Windows\\CurrentVersion\\Uninstall\\Google Chrome$
  </key>
  <name>DisplayName</name>
</registry_object>
```

Architecture

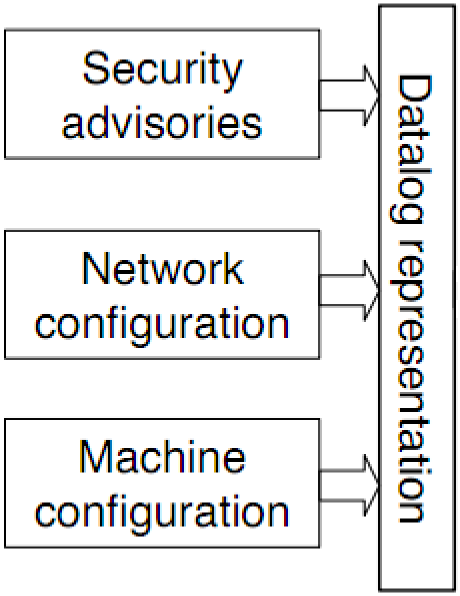


Perhaps something like OVAL

e.g., Firewall config

e.g., Anti-virus config, patch history

Architecture



Convert this information into Datalog. This is a manual step.

Background: Datalog

Atoms are of the form:

$p(X_1, X_2, \ldots, X_n)$

Predicate → p

$X_1, \ldots, X_n$: Variables, Predicates or Constants

- Variables are capitalized
- Predicates and Constants are lower case

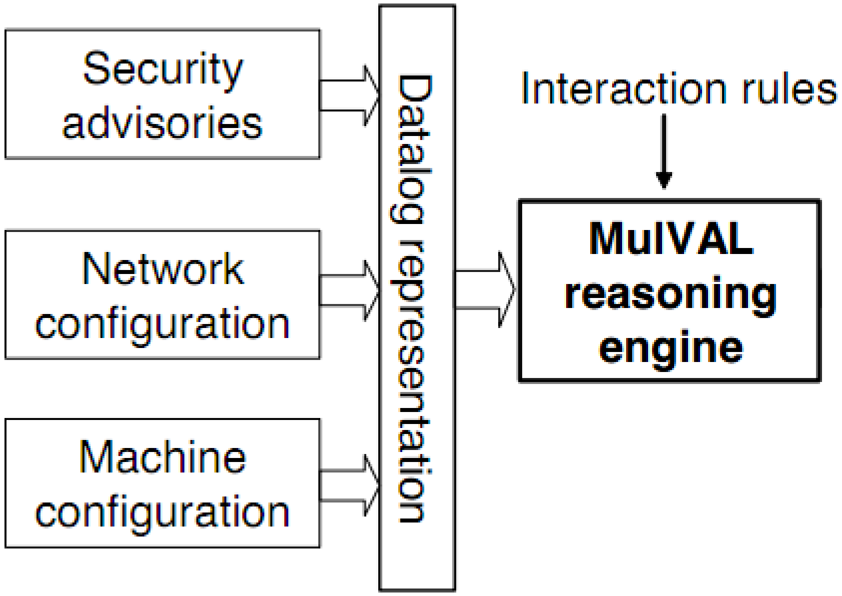
Background: Datalog

Datalog Rules:

$$H \text{ :- } B_1, B_2, \ldots, B_n$$

- H is an atom and B_1 through B_n are literals (atoms).
- The symbol :- can be read as "if"
- More precisely stated: "The head is true if the body is true."
- A Datalog program is a collection of rules

Architecture



MulVAL evaluates interaction rules on input facts.

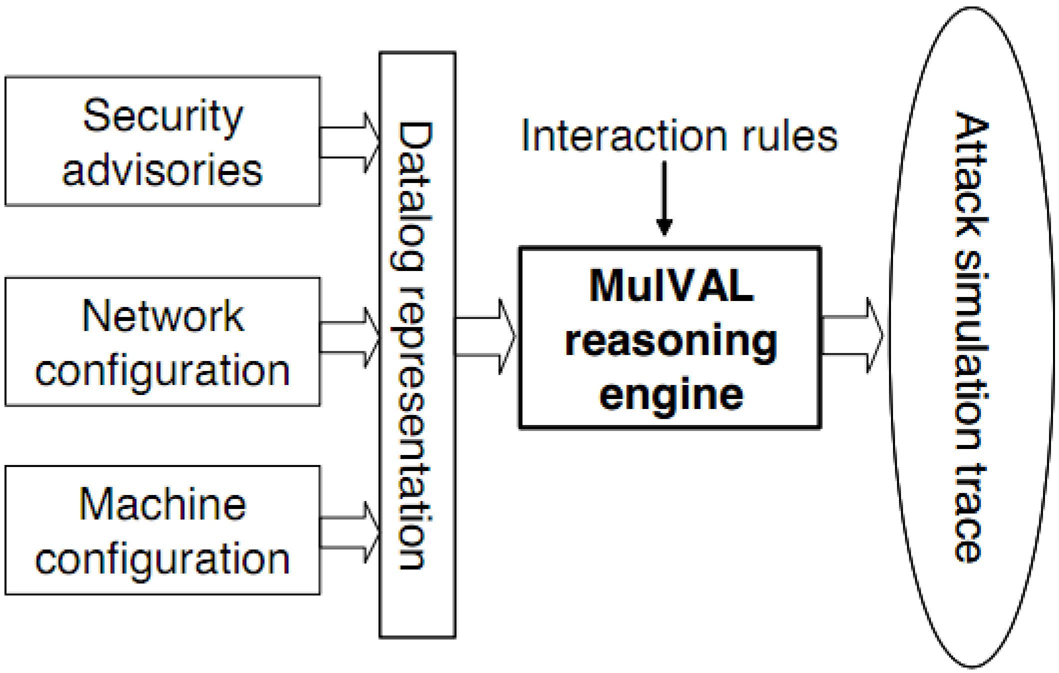
MulVAL can automatically identify/derive security vulnerabilities, assuming it has been provided the correct inputs in Datalog format.

Interaction rule

```
execCode(Attacker, Host, User) :-
  networkService(Host, Program,
                 Protocol, Port, User),
  vulExists(Host, VulID, Program,
            remoteExploit, privEscalation),
  netAccess(Attacker, Host, Protocol, Port).
```

- If an attacker can execute code on a host
- The host had a listening network service AND
- The program had a vulnerability AND
- The attacker had public access to the service.

Architecture



MulVAL was modified to perform a "trace" when doing a DFS of the graph in addition to providing a simple "yes" or "no" to a vulnerability query.

Modifying interaction rules

PENNSTATE 1855

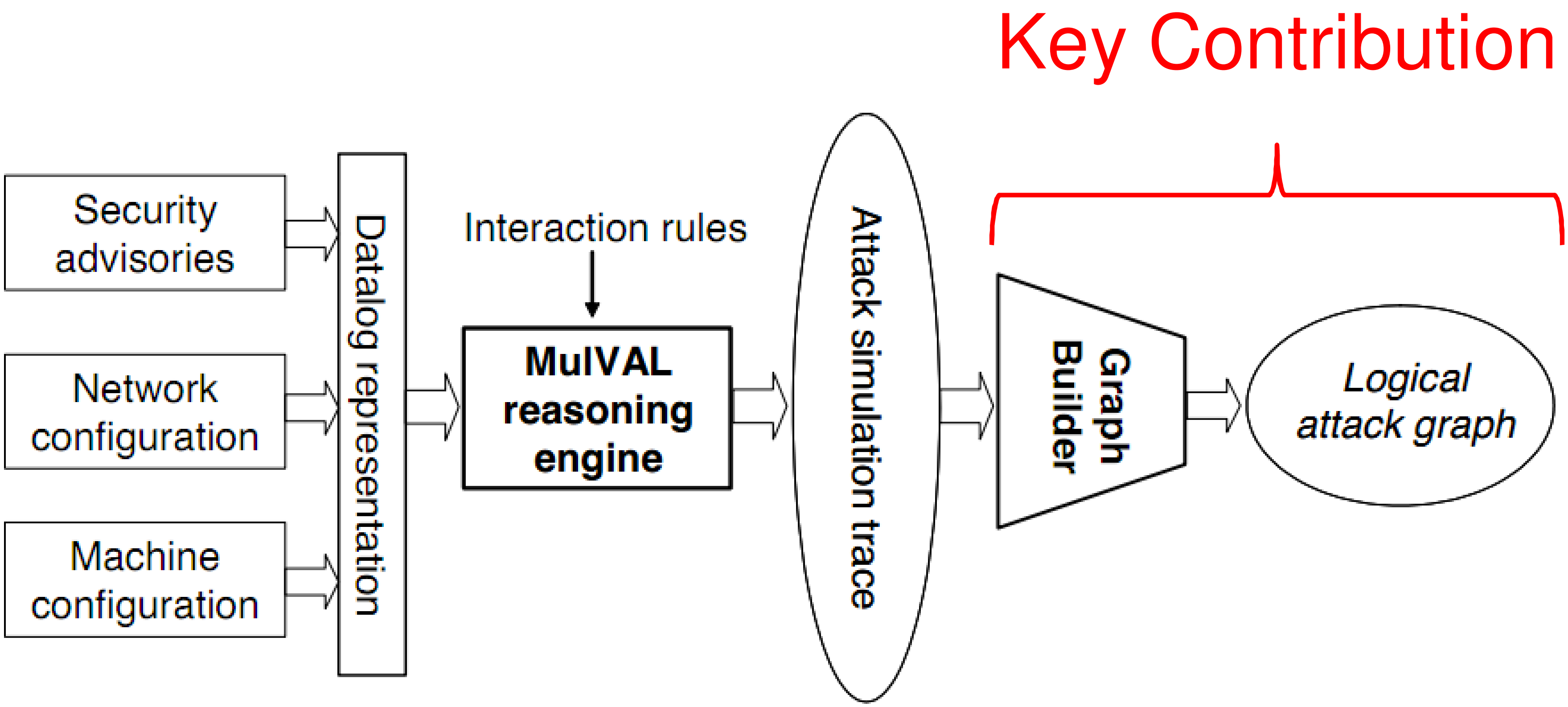
Before

```
execCode(Attacker, Host, User) :-
  networkService(Host, Program,
                  Protocol, Port, User),
  vulExists(Host, VulID, Program,
            remoteExploit, privEscalation),
  netAccess(Attacker, Host, Protocol, Port).
```

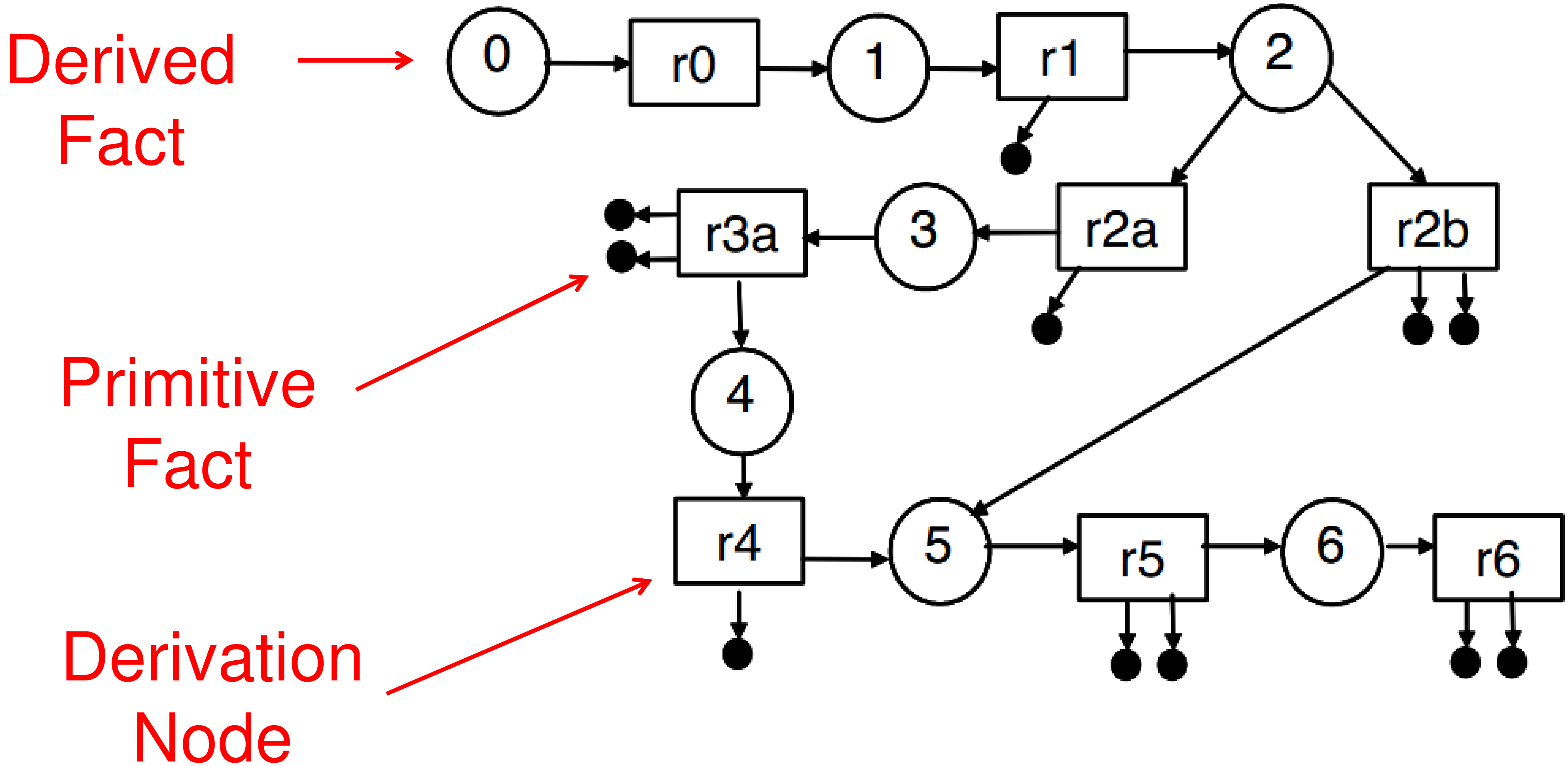
After

```
execCode(Attacker, Host, User) :-
  networkService(Host, Program,
                  Protocol, Port, User),
  vulExists(Host, VulID, Program,
            remoteExploit, privEscalation),
  netAccess(Attacker, Host, Protocol, Port),
  assert_trace(because(
 'remote exploit of a server program',
   execCode(Attacker, Host, User),
     [networkService(Host, Program,
                      Protocol, Port, User),
      vulExists(Host, VulID, Program,
                remoteExploit, privEscalation),
      netAccess(Attacker, Host,
                Protocol, Port)])).
```

Architecture



Logical Attack Graph



Constructing the graph

```
execCode(Attacker, Host, User) :-
  networkService(Host, Program,
                 Protocol, Port, User),
  vulExists(Host, VulID, Program,
           remoteExploit, privEscalation),
  netAccess(Attacker, Host, Protocol, Port),
  assert_trace(because(
 'remote exploit of a server program',
   execCode(Attacker, Host, User),
     [networkService(Host, Program,
                     Protocol, Port, User),
      vulExists(Host, VulID, Program,
               remoteExploit, privEscalation),
      netAccess(Attacker, Host,
                Protocol, Port)])).
```

Definition 2. Attack simulation trace.

$$
\begin{aligned}
\mathit{TraceStep} &::= \mathbf{because}(\mathit{interactionRule}, \\
&\qquad\qquad \mathit{Fact}, \mathit{Conjunct}) \\
\mathit{Fact} &::= \mathit{predicate}(\mathit{list\ of\ constant}) \\
\mathit{Conjunct} &::= [\mathit{list\ of\ Fact}]
\end{aligned}
$$

Constructing the graph

- Every TraceStep term becomes a derivation node in the attack graph.
- The Fact field in the trace step becomes the node's parent
- The Conjunct field becomes its children.
- Iteratively repeat until we've exhausted our interaction rules.

Performance Results

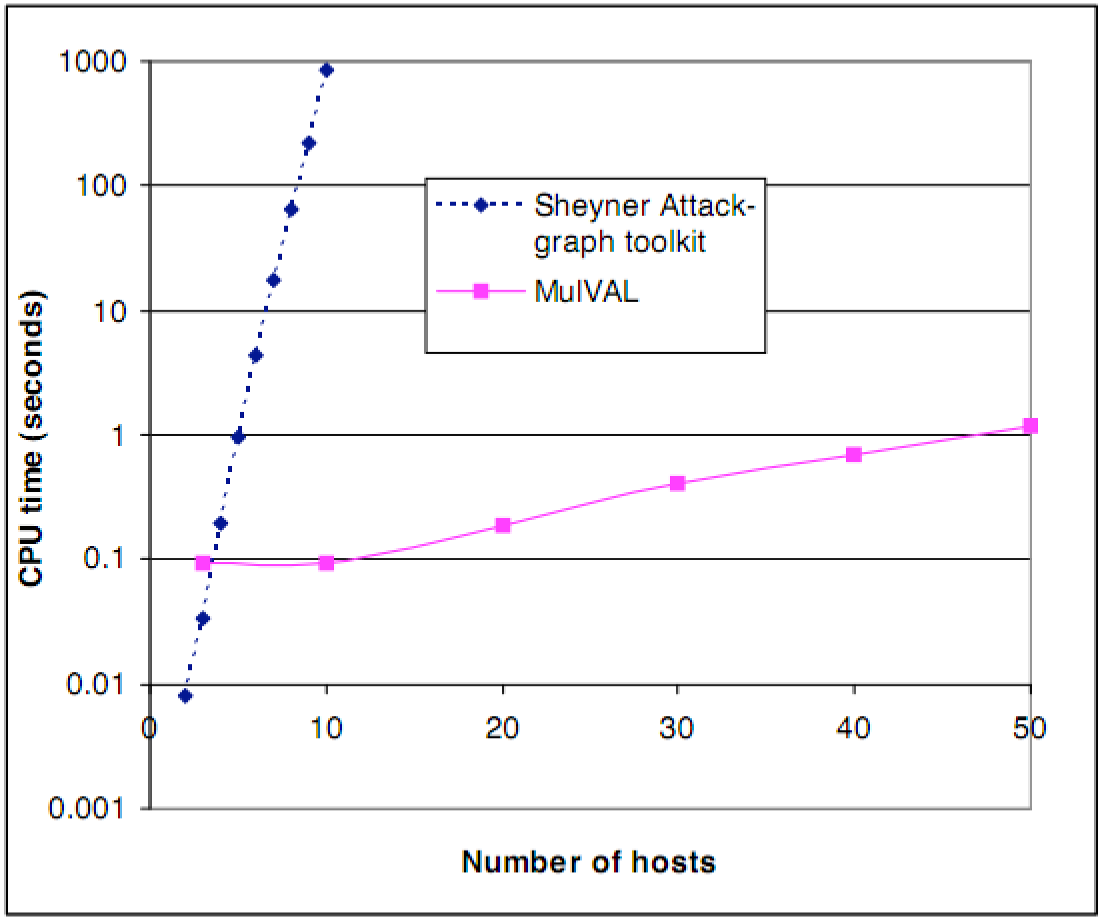


Figure 14: Graph generation CPU time compared to Sheyner attack graph toolkit. Fully connected network and 5 vulnerabilities per host.

Performance results compared with the closest competitor