A Scalable Approach to Attack Graph Generation*

Jim Owens
Advisor: Chris Lynch

*by Xinming Ou, Wayne F. Boyer & Miles A. McQueen, published at ACM's CCS '06
Overview

- Motivation & background
- Logical attack graphs (2006)
  - Algorithms
  - Complexity analysis
  - Experimental results
- Conclusions & future work
Motivation

• Growing complexity of network security problem suggests need for automated security tools
  - Incremental security improvements require multi-stage, multi-host attacks to reach given targets
  - Profit motive leads attackers to try harder
  - Many enterprise networks comprise 1,000's or even 10,000's of hosts
  - Large, complex networks are difficult for humans to understand and defend
Motivation

• *Example*: TJX compromise in 2003-2006 exposed data on 94 million+ credit cards
  - Original attacks targeted WEP- or unsecured wireless access points via *war driving*
  - Once inside, attackers used 'sniffers,' SQL injection attacks, other means to extend compromises
  - Later, attackers installed a VPN between a TJX transaction processing server and an attack server to capture data “as it was being processed”
Background

- *Attack graphs* (or *trees*) can be used to model attacks and analyze system security
- The goal or target is the root node, with initial points of attack represented as leaf nodes
- Edges define transitions between attack stages
- Both AND and OR nodes can be used
- Attack graphs can effectively model multi-host, multi-stage attacks
Sample attack tree (Schneier)
Review of early work

• In March 2005, MIT's Lincoln Lab reviewed 18 research papers on attack graphs for USAF
  − Scalability a primary concern; the best systems scaled as $O(N^6)$, limiting application to small networks only
  − Most systems used manual entry of configuration data and hand-drawn attack graphs
  − Network reachability analysis also identified as an important weakness
Review of early work

• In 2002 Sheyner, et al, proposed an automated system using a FSM & model checking
  - Required hand-generated network input
  - Graph nodes consist of Boolean variables, representing the entire network state
  - Graph building/analysis for 4 hosts/4 vulns: 5 secs
  - Graph building/analysis for 5 hosts/8 vulns: 2 hours
  - Raised serious questions on suitability of model-checking approaches
Logic-based attack graphs

• In 2006, Xinming Ou and two researchers from Idaho National Laboratory introduce logic-based attack graphs
  - Machine-generated graphs based on logic programming techniques
  - Vulnerability analysis and graph generation scales at $O(N^2 \log(N))$
Logic-based attack graphs

- Require that an attacker's potential privileges be expressed as a propositional formula, in terms of network/host configuration parameters
  - Nodes represent logical statements, encoding some aspect of host/network configuration
  - Edges specify causality relations between configuration parameters and an attacker's potential privileges
Logic-based attack graphs

- Vulnerabilities identified using MulVAL*, a reasoning engine based on XSB Prolog and Datalog interaction rules
  - Configuration information represented as Datalog tuples
  - Attack techniques and OS semantics represented as Datalog interaction rules

*Multi-host, Multi-stage Vulnerability Analysis Language
Logic-based attack graphs

- Sample Datalog interaction rule:

\[
\text{execCode} (\text{Attacker}, \text{Host}, \text{User}) :\neg \\
\text{networkService} (\text{Host}, \text{Program}, \\
\text{Protocol}, \text{Port}, \text{User}), \\
\text{vulExists} (\text{Host}, \text{VulID}, \text{Program}, \\
\text{remoteExploit}, \text{privEscalation}), \\
\text{netAccess} (\text{Attacker}, \text{Host}, \text{Protocol}, \\
\text{Port})
\]
Logical attack graph generator
Example network

CAN-2002-0392

CVE-2003-0252
Logical attack graph
Tree representation (1)

<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)
Tree representation (2)

<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)

   -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

   <r2b>Rule15: NFS shell
   |   []-hacl(webServer,fileServer,rpc,100003)
   |   []-nfsExportInfo(fileServer,/export,write,webServer)
   |   |--execCode(attacker,webServer,apache) ==> <5>
Algorithms

- MulVAL engine yields yes/no answers on exploitable vulnerabilities using interaction rules
- Attack simulation traces also produced for each fully-satisfied derivation rule
- Simulation traces are written to file, and later processed into attack graphs using the “map” template in the C++ standard library
Algorithms

• Attack simulation trace:

  \[ \text{TraceStep} ::= \text{because}(\text{interactionRule}, \text{Fact}, \text{Conjunct}) \]

  \[ \text{Fact} ::= \text{predicate}(\text{list of constant}) \]

  \[ \text{Conjunct} ::= (\text{list of Fact}) \]

  – Each \text{TraceStep} term becomes a derivation node in the attack graph

  – \text{Fact} field becomes the node's parent

  – \text{Conjunct} field becomes its children.
Complexity analysis

• Complexity of computing attack trace

Theorem 1: Evaluating MulVAL interaction rules against configuration tuples representing $N$ hosts takes $O(N^2)$ derivation steps.

Proof: For fixed Datalog programs, running time is dominated by rules with the max number of body-variable instantiations. The rule with the highest number of such instantiations—netAccess—can have instantiations for every host with every other host on the network: $N^2$. 
Complexity analysis

\text{netAccess}(\text{Attacker}, \ H2, \ \text{Protocol}, \ Port) \ :-
\text{execCode}(\text{Attacker}, \ H1, \ User),
\text{hacl}(H1, \ H2, \ \text{Protocol}, \ Port)

1) Compute all hosts on which attacker can execCode
2) Compute all \( H1 \)s and \( H2 \)s between which network access is possible
3) Finally, compute \text{netAccess} by matching results of the two sub-goals, which have been written to tables (XSB claims very efficient pattern-matching)
Complexity analysis

• Complexity of computing attack trace

Every trace step is produced by one derivation step in Datalog evaluation, so based on Theorem 1 we also have...

Corollary 1: The number of trace-step terms produced in attack simulations is $O(N^2)$. 
Complexity analysis

• Complexity of graph building

Theorem 2: The logical attack graph for a network with \( N \) hosts has a size at most \( O(N^2) \).

Proof: There is a 1:1 correspondence between TraceStep terms and derivation nodes. If there are \( D \) trace steps, then there are \( D \) derivation nodes in the graph. If there are at most \( m \) preconditions for a rule, the number of edges in the graph is at most \( mD \), and the maximum number of fact nodes is \( mD+1 \). By Corollary 1, \( D \) is \( O(N^2) \), as is \( mD+1 \).
Complexity analysis

• Complexity of graph building

Theorem 3: The graph building algorithm takes time $O(N^2 \log(N))$ to complete, where $N$ is the number of hosts in the network.

Proof: The algorithm loops through all TraceStep terms, which by Corollary 1 is $O(N^2)$. In each iteration, the algorithm creates a derivation node for the term and makes links from its parent and to its children. Only table look ups are not constant time; these take $\log(n)$ time.
Experimental results

Figure 9: Graph generation CPU usage as a function of network size for several network topologies.
Experimental results

Figure 13: Graph generation CPU time for a fully connected network and number of vulnerabilities per host varying from 1 to 100.
Conclusions

• Logical attack graphs
  - Directly illustrate logical dependencies between attack goals and configuration information
  - Show dramatically improved scaling over earlier approaches
  - Have size polynomial to the network being analyzed
Future work

• Develop algorithm to eliminate useless loops in attack graphs

• Create custom map library to make table lookups more efficient during graph building phase

• Automate the generation of Datalog tuples from vulnerability database (NVD) and network and machine configuration data