T1C1 10:00 Proceedings of the 2001 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY, 5-6 June, 2001

Modeling Internet Attacks

T. Tidwell, R. Larson, K. Fitch and J. Hale*

***Abstract*—As the frequency and complexity Internet attacks increase, systems administrators need more sophisticated tools to warn and direct their responses. The foundation for any such effort is a coherent model of exploits and vulnerabilities that is rich enough to capture the behavior and composition of multi-stage attacks. This paper describes an enhanced attack tree model of Internet attacks, and a companion specification language for expressing aggregate attack behaviors and modalities. A distributed attack notification and visualization system is briefly described that uses the model as a common representation for incidents captured by Intrusion Detection Systems (IDSs).**

***Index Terms*—attack tree, vulnerability analysis, multi-stage and coordinated attacks.**

I. INTRODUCTION

THE increasing frequency and complexity of Internet attacks has raised the level of sophistication required by systems administrators to effectively cope with script kiddies and more sophisticated hackers. Multi-stage attacks can be orchestrated to strike highly protected targets, to coordinate waves of scripted exploits and/or to conceal the true origin of an attack. Unfortunately, correlating information from such incidents is a difficult task exacerbated by heterogeneity, interoperability and policy issues, made impossible by the lack of a suitable model for Internet attacks.

Identifying coordinated attacks in progress is still a black art that relies on luck more than scientific methodology. Linking one exploit with another, potentially across a global network of heterogeneous and federated systems, requires perseverance, cooperation, timeliness and a chain of recognizable evidence that must survive hacker cover-ups, purged logfiles and system administrator oversight. Advanced Intrusion Detection Systems excel at collecting information across heterogeneous networks [4,6,13,15,16]. Yet most still regard coordinated attacks as unrelated collections of intrusions. Next generation IDSs must cooperate with each other and provide intuitive feedback for users to recognize a pattern of attack from seemingly unrelated events [2,3,18].

This paper presents an Internet attack model capable of capturing composite attacks and briefly describes a distributed attack notification and visualization system. The model extends the attack tree concept [14] with parameters, precondition and postcondition assertions, and other features inspired by modern programming languages. Dual specification languages are used to express exploits and network characteristics. Together, they permit systematic visualization, vulnerability assessment and attack prediction by a centralized monitor observing activity in a network.

*To whom correspondence should be addressed (email: john-hale@utulsa.edu).

II. BACKGROUND AND RELATED WORK

To this point, most of the research in the area of attack modeling has focused on classifying and categorizing exploits and vulnerabilities [1,7,8,9,10]. [5] proposes a comprehensive taxonomy of Internet attacks based on effect and intent, and presents statistics relating the frequency of incidents in each category as reported to CERT from 1989-1995. Mitre's CVE database creates a common namespace for all vulnerabilities and exploits [11]. The NIST I-cat project provides a database interface over the web that allows users to browse and search for exploits by platform, intent and other criteria [12]. Such taxonomies and characterizations provide useful insights into the nature of computer attacks, but fail to formally express their composable properties.

In terms of modeling the behavior and effect of exploits, attack trees offer a goal-oriented perspective that facilitates expression of multi-stage attacks [14]. Attack trees in their simplest form assert subgoals for achieving the goal set forth by an attack node. Attack nodes can be grouped into 'AND' or 'OR' sequences to capture conjunctive and disjunctive attack conditions, respectively. Nodes can be weighted to reflect the likelihood of successfully mounting an attack.

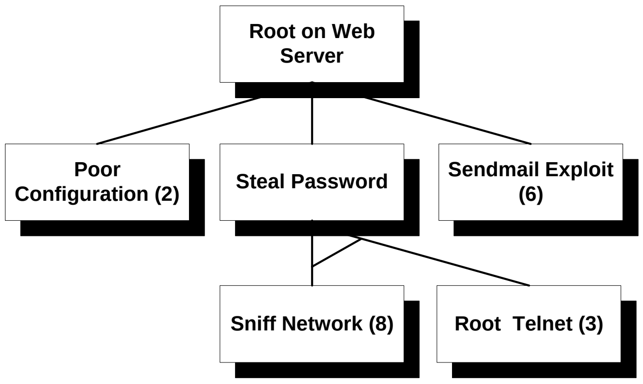

Fig. 1 Web Server Attack Tree

In Figure 1 an attacker is seeking to gain root access to a web server. There are 3 possible methods presented for obtaining root access to this particular machine; 1) poor configuration of the web server, 2) stealing the root password, or 3) executing a sendmail exploit. The 'AND' nodes in this example express that stealing a password may be accomplished if a sniffer is in place during a root telnet session. Attack tree nodes can be weighted to represent the likelihood of success in achieving a goal. The tree in Figure 1 uses a likelihood scale of 1 (least likely) to 10 (most likely).

```
goal ::=  attack name (params)
        {props vars preconds subgoals postconds}
params ::= param (, param)*
param ::=   type ID
props ::=   (prop)*
prop ::=    property  type ID = val  ;
vars ::=     (var)*
var ::=      declare type ID ;
preconds ::= preconditions { orExpr }
orExpr ::=  andExpr ( | andExpr )*
andExpr ::= notExpr ( & notExpr )*
notExpr ::=  grpExpr | ~ grpExpr
grpExpr ::=  atomicRule | ( orExpr )
atomicRule ::=  val boolOp val
boolOp ::=  contains | instanceof | = | > | >=
            | < | <= | !=
subgoals ::= subgoals { ( andSubgoals )*  }
andSubgoals ::=   (subgoal ( , subgoal )*)
subgoal ::=   name ( args )
args ::=      val ( , val )*
postconds ::=postconditions {  (stmt )*  }
stmt ::=   name := val ;
val ::=     literal | name
name ::=  ID ( . ID )*
literal ::=  version_literal | integer_literal
           | boolean_literal  | date_literal
           | string_literal
type ::=  version | integer | boolean
                | date | string | name
```

Leaf node weights can be computed or hand-specified, while internal branch weights are derived from the leaf node weights. (Poor configuration is rated at a 2 as an unlikely occurrence.)

Fig. 2 Attack Specification Language BNF.

III. Attack Modeling

Parametric extensions to attack trees yield a flexible model for capturing coordinated and multi-stage attacks. Each attack tree specification template is defined with a unique identifier, and a list of system element parameters. Templates contain descriptive properties, preconditions, subgoals and postconditions. Abstract and concrete goal-oriented attack behavior can be expressed within the model; system element parameters link attack specification templates to references in subgoals, preconditions and postconditions. Figure 2 contains the BNF specification for the parametric attack tree modeling language.

Properties in the specification language perform an auxiliary function in the model, and can be used to express arbitrary exploit characteristics. For example, a common property is description, which can be bound to a string literal that textually describes the exploit. CVElink and version are also common properties that link attack specifications to the Mitre CVE database [11] and express the version of the attack specification, respectively. Impact and likelihood of success weights may also be stored as attack properties for use by vulnerability assessment algorithms and heuristics.

Preconditions express system environment or configuration properties that may facilitate or inhibit the successful execution of an attack. Arbitrarily complex first order predicate logic and boolean phrases may be articulated to precisely describe enabling conditions for system compromise. Precondition expressions may contain references to local system variables or parameters.

Subgoals differ from preconditions in that they represent antecedent objectives of system intrusions or compromises. I.e., hackers search for favorable preconditions conducive to the execution of particular exploits, while they must actively fulfill attack subgoals in the pursuit of their ultimate goals. Moreover, subgoal references must be attack node references applied to system arguments within a strict "AND/OR" hierarchical framework.

The format of a subgoal reference closely resembles a function call in a conventional programming language (the subgoal identifier applied to a list of arguments). As in a function, the argument values are bound to formal parameters in the subgoal definition (specification) upon evaluation.

Postconditions identify state changes in systems and environments. These may alter the characteristics of network and host elements (e.g. effecting a version rollback attack or the placement of a logic bomb within software) or they may express the viability or satisfaction of concrete or abstract attack nodes, respectively.

Specifiers may use parameters to model attacks applicable to a class of devices or systems. System element and attack values and variables are bound to parameters in attack templates producing attack nodes representing intrusion events. The parametric attack modeling language uses a hierarchical namespace for system elements comprising the model's type system. The namespace reflects a taxonomical organization of network, system and host elements. For example, System.OS.Linux.Version is an element of the namespace tree that maps to all legal version numbers of the operating system Linux. These elements can be used in the left hand side of an assignment or comparison expression or to type or cast a literal value.

Boolean operators use the namespace to create ownership and inheritance relationships between system elements. The contains operator checks to see if an element contains a field with a given name, while the instanceof operator checks if an element is an instance of a given type. Operands can be literals, parameters or locally declared variables.

Abstraction plays a critical role in modeling composite attacks. While concrete nodes are identified with specific exploits, abstract nodes are used to serve as placeholders in an attack tree for conceptual objectives such as execute_arbitrary() or halt_system(). Abstract nodes may be linked to concrete nodes or to each other to express the general effect of an attack node.

The principal use of the attack modeling language described here is to systematically express compositional modes of computer and network attack. In practice, a complete attack tree defined for a host would include specific techniques and vulnerabilities that would regard potential exploits satisfied by system and subsystem properties. E.g., an attack template might only apply to a machine running a specific version of Linux or a Windows NT server configured in a particular manner. Thus, networks, hosts and their relevant characteristics must also be modeled faithfully to provide the backdrop for vulnerability assessment. A network model and specification language has been designed for this purpose. Network specifications can be used to guide the use of attack trees in vulnerability assessment. I.e., attack trees must be constructed and then pruned to model precisely those exploits associated with a particular system.

System specifications represent concrete structures such as computer networks or hosts. Figure 3 contains a BNF grammar for a system specification language, which consists of two parts: a *type definition* and an *instance*. The *type definition* is a set of entity type declarations, similar to complex data types in modern programming languages. Each type represents a class of entity in a given domain, such as hardware (e.g. a router), software (e.g. an operating system or application), or even human (e.g. employees) and paper (e.g. files). (As a result the model facilitates the analysis of social engineering threats as well as those from cyber attack.)

Each type definition consists of a name and a list of declared properties. Properties can be primitive types; version, integer, date, etc. They can also be references to other entities (entity types), lists of a given type, or internal entities. Properties describe the defined entity. Default property values can be specified in type definitions.

The system specification language uses a hierarchical type system. Thus types can inherent properties of other type definitions by using the extends keyword. Properties of the parent type are implicitly bestowed upon the child type but can be overridden. Type names consist of an identifier and a namespace.

declUnit ::= (typeDecl | objDecl)*
typeDecl ::= **type** name **{** propDecl* **}**
propDecl ::= type ID (**=** val)$_{opt}$ **;**
name ::= ID (**.** ID)*
type ::= primType | listType | refType
primType ::= **version** | **integer** | **boolean** |
date | **string**
listType ::= type **list**
objDecl ::= name name **{** propAssign* **}**
propAssign ::= ID **=** val **;**
val ::= literal | name
literal ::= primLiteral | listLiteral | objLiteral
primLiteral ::= boolean_literal | integer_literal
| version_literal | string_literal
| date_literal
listLiteral ::= **{** val (**,** val)* **}**
objLiteral ::= name objBody

Fig. 3 System Specification Language BNF.

Namespaces are used to aid in the organization and separation of entity classes. Namespace elements are syntactically similar to Java packages. Entity class standard names such as 'Machine' and 'Application' are defined to provide a common naming convention for system specifications.

The second part of the system specification language, the *instance*, uses declared types to model an actual system, network or host. Entities are declared with a type and optional name, much like a variable. The body of the entity declaration establishes properties of the entity defined in the type declaration. Properties can be entities themselves, allowing complex systems and subsystems to be modeled.

IV. EXAMPLE

This section presents a simple attack scenario modeled with the attack and system specification languages, respectively. As shown in Figure 4, the system modeled is a simple network containing two hosts of particular interest; dante.utulsa.edu and virgil.utulsa.edu.

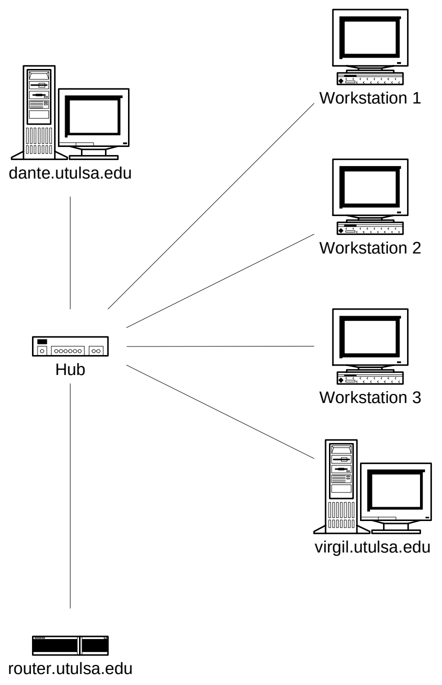

Fig. 4 Network Topology.

The multi-stage exploit modeled in this scenario affects two hosts (dante and virgil), using one to attack the other. Figure 5 presents a thumbnail sketch of the hosts involved in the attack.

`dante.utulsa.edu` (240.1.1.1)
OS: Sun Solaris 7.0 x86
Services: MySQL 3.23.9
in.telnetd

`virgil.utulsa.edu` (240.1.1.1)
OS: Redhat Linux 6.1 i386
Services: BIND (named) version 8.2

Fig. 5 Host Profiles.

The attack scenario on this network plays out as follows:

1. The attacker first identifies virgil.utulsa.edu as a linux system running a vulnerable version of *named*. She performs a buffer overflow exploiting the ‘nxt bug’ vulnerability (CVE-1999-0833 – Bugtraq ID 788) to obtain access to virgil’s root account.

2. Next the attacker sets up a packet sniffer on virgil in order to capture all traffic on the local (un-switched) network. Eventually, she is able to capture a clear-text telnet login session from a local workstation to dante and extracts username and password information from it. She now has access to a valid user account on dante.

3. The attacker then logs onto dante with the username and password she sniffed. From the account, she utilizes a vulnerability in MySQL (SELECT statement local buffer overflow exploit - Bugtraq ID 2262) to obtain root access on dante.

This attack sequence employs a local buffer overflow in MySQL, and a BIND vulnerability – one of the most prevalent network vulnerabilities as recognized by the SANS Institute [17]. Abbreviated profiles of both the BIND_nxt and the MySQL_SELECT vulnerabilities are given in Figure 6.

BIND nxt Vulnerability
[http://www.securityfocus.com/bid/788]
Date: November 10, 1999
Bugtraq ID: 788
CVE name: CVE-1999-0833

MySQL SELECT Vulnerability
[http://www.securityfocus.com/bid/2262]
Date: January 18, 2001
Bugtraq ID: 2262
CVE name: CVE-MAP-NOMATCH

Fig. 6 Attack Profiles.

Each of these vulnerabilities is modeled as an attack template with the attack specification language. Figures 7 and 8 show attack specifications expressed as templates and stored within the files ExploitBIND_nxt_Vulnerability.atk and ExploitMySQL_SELECT_Vulnerability.atk, respectively.

The attack specification for the BIND_nxt exploit is parameterized by Service and expresses preconditions on that service identifying it as named and constraining the version to between v8.2 and v8.2.2. The single subgoal for this exploit is gaining access to port 53.

The attack specification for the mySQL exploit is parameterized by Application and its preconditions express the requirement that the application be an instance of MySQL, constraining the version to between v3.22.26 and v3.23.9. The only subgoal here is being able to execute the application.

```
/* ExploitBIND_nxt_Vulnerability
 * Author: Kenneth Fitch
 * Date: 2001 March 22
 * Src: SANS Top 10, www.securityfocus.com,
 * www.cert.org/advisories/CA-1999-14.html
 * Notes:
 *  BIND weaknesses #1 Internet security
 *  threat according to SANS
 *  www.sans.org/topten.htm>
 */

attack service.bind.ExploitBIND_nxt_Vulnerability
(Service s)
{
  property string CVE_ID = "CVE-1999-0833";
  property string CERT_advisory = "CA-1999-14
  Multiple Vulnerabilities in BIND";
  property string BugTraqID = "788";

  preconditions
  { s instanceof named_Service &
    s.version >= v8.2 & s.version < v8.2.2
  }

  postconditions{}

  subgoals
  { ( ConnectToHost( s.host, 53 ) ) }
}
```

Fig. 7 ExploitBIND_nxt_Vulnerability.atk

Stylistically, liberal use of property fields and comments makes the specification much more readable (and in the specific case of properties, searchable). Postconditions in these specifications have been omitted: They are not needed to construct the attack tree, but may be useful and even necessary to perform vulnerability assessment and attack prediction.

```
/* ExploitMySQL_SELECT_Vulnerability
 * Author: Kenneth Fitch
 * Date: 2001 March 22
 * Src: http://www.securityfocus.com/bid/2262
 */

attack
application.mysql.ExploitMySQL_SELECT_Vulnerabilit
y
(Application a)
{
  property string CVE_ID = "CVE-MAP-NOMATCH";
  property string BugTraqID = "2262";

  preconditions
  { a instanceof MySQL_Application &
    a.version >= v3.22.26 & a.version <= v3.23.9
  }

  postconditions {}

  subgoals
  { ( ExecuteProgram(a) ) }
}
```

Fig. 8 ExploitMySQL_SELECT_Vulnerability.atk.

Figure 9 illustrates the full attack tree for compromising dante via virgil. The tree can be divided into two subtrees; one that models the attack on virgil (contained in the lower box in Figure 9) and one that models the attack on dante (contained in the upper box in Figure 9). The tree exhibits a uniform chain structure except for the subgoals of

ExploitMySQL_Select_Vulnerability(dante), which comprise two 'AND' nodes – CompromiseAccount(dante) and ExecuteProgram(dante.MySQL) – both of which are required to trigger the exploit. Note the use of abstract nodes as mortar to connect concrete nodes in the tree. This is a common technique, as is layering abstract nodes to manage complexity and to model non-exploit (authorized) behavior.

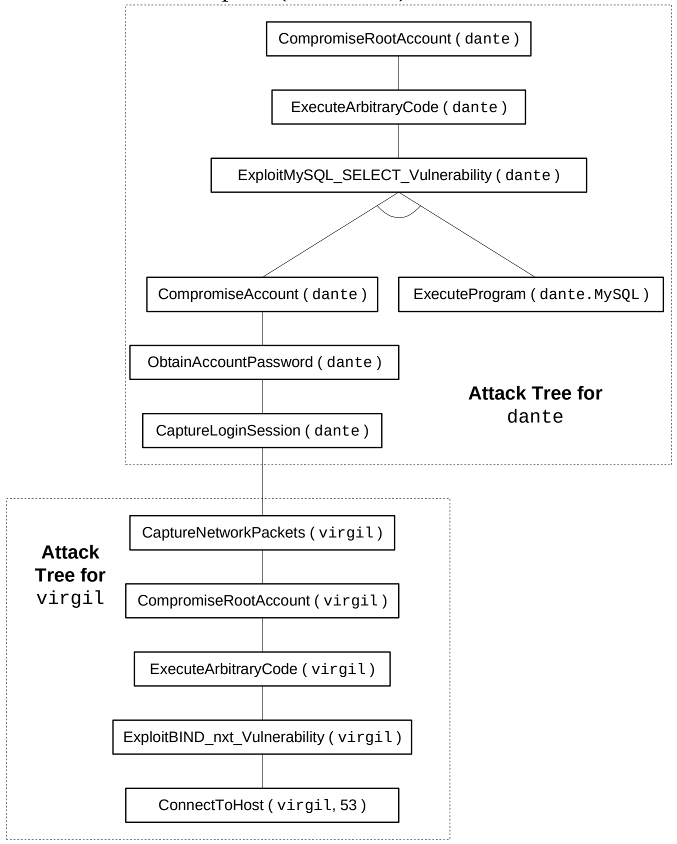
CompromiseRootAccount (dante)
ExecuteArbitraryCode (dante)
ExploitMySQL_SELECT_Vulnerability (dante)
CompromiseAccount (dante)
ExecuteProgram (dante.MySQL)
ObtainAccountPassword (dante)
CaptureLoginSession (dante)
Attack Tree for dante
Attack Tree for virgil
CaptureNetworkPackets (virgil)
CompromiseRootAccount (virgil)
ExecuteArbitraryCode (virgil)
ExploitBIND_nxt_Vulnerability (virgil)
ConnectToHost (virgil, 53)

Fig. 9 MYSQL_SELECT Attack Tree.

V. ATTACK NOTIFICATION SYSTEM

The Attack Visualization System derives its functionality through the interaction of four distinct entities: the Network and Attack Model Databases, the Attack Notification Service, and the Attack Monitor. Each component plays an integral role in helping systems administrators and ISSOs identify and extinguish multi-stage network attacks.

The Network and Attack Model Databases are populated from the system and attack specification languages, respectively. Special compilers parse expressions of each language, generating the database update commands necessary to map the model into a relational structure. The Attack Model Database serves as a centralized repository of exploit knowledge remotely accessible by a large population of network administrators and ISSOs. This database contains a relational view of individual "templates" that are self-contained goal-oriented descriptions of an attack. On the other hand, the Network Specification Database is intended to provide an integrated view of an individual enterprise network, and as such may only be accessible by a handful of administrators and staff within a single enterprise. Individual network elements and their properties will be stored to provide easily accessible information about the current network state. To further allow the Network Specification Database to represent current network conditions it can be updated on-the-fly by network scanning or attack monitoring software.

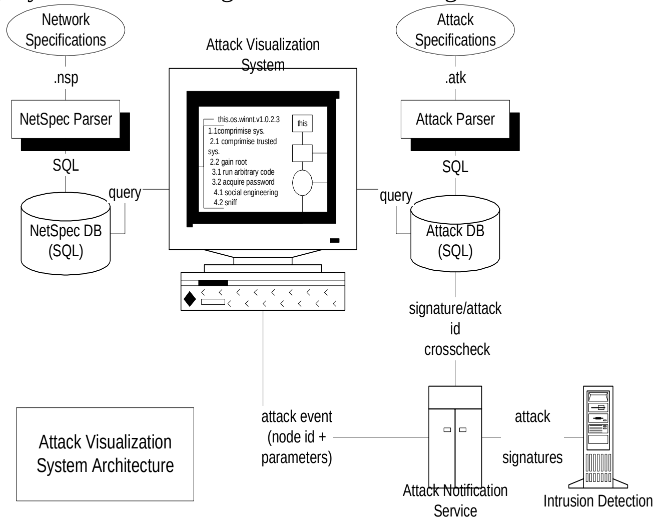

Fig. 10 Attack Visualization System and Notification Architecture.

While one of the primary uses of the models and visualization system is to conduct vulnerability assessments, the architecture can be extended to provide real-time attack notification and monitoring services. By strategically placing Attack Notification Service software on hosts in a network, the central monitor can be informed of hostile activities, rendering them in real-time on a graphic overlay of the network under attack. Attack Notification Service software must be linked with an Intrusion Detection System to correlate audit data with attack signatures. Once an attack is discovered and identified by an IDS, the Attack Notification Service retrieves the correct attack template from the Attack Model Database and uses it to instantiate an attack node, binding formal parameters to arguments along the way. The attack node is then shipped to the centralized monitor where it can be analyzed and potentially placed within an attack tree to capture its role in a multi-stage or coordinated attack.

The Attack Monitor contains the visualization and analytical functions that implement and render vulnerability assessment, attack predication and real-time attack notification. It derives its functionality from interaction with the other components in the system. The Network Specification Database provides a model of the system under observation. The Attack Notification Service provides warnings and data regarding attacks in-progress. The monitor can update the Network Specification Database as events (hostile or not) effect state changes on hosts and networks.

The visualization component of the monitor's software architecture leans heavily on the object-oriented representation of system and exploit elements. Adopting a 'model/view' approach to graphical representation, the visualization system consists of intelligent rendering methods placed within each model element class. These methods must be able to intuitively display current network information and attacks, and visually distinguish between contrasting logical states. They must also be aware of their context to provide a suitable graphical representation that is appropriately scaled and located within the current view. Moreover, graphical elements in the system must be interactive so that when users wish to focus their attention on particular objects, they can easily manipulate the view to accommodate their interests.

Analytical functions within the monitor serve several purposes; vulnerability assessment, tree construction and attack prediction. Each of these functions executes in either online or offline mode. Online mode engages the Attack Notification Services to observe and evaluate incidents in real-time over a network. Offline mode disengages the Attack Notification Service so that simulations and hypothetical analyses may be conducted.

Vulnerability assessment functions construct an attack tree with a root objective targeting a specific system element in a network specification. The attack tree is constructed in a top-down fashion by chaining attack templates that match vulnerabilities found within the active network specification. Weights reflecting likelihood of attempt or success may be used to prune branches that do not rise above a minimum threat threshold.

Tree construction functions model on-going attacks. Attack Notification Services transmit attack nodes representing actual attacks discovered by IDSs. Attack trees may also be constructed from a simulation mode that replaces the Attack Notification Service with a local attack node generation software component. In either case, the monitor collects attack nodes and uses heuristic methods and abstract nodes to construct a forest of attack trees linking recent and on-going attacks within a network. Attack tree nodes are linked to the system elements they affect, creating an intuitive navigation scheme for user inspection.

Disconnected attack trees may be the result of unrelated hostilities, or undiscovered incidents. Analytical functions within the monitor can query the Attack Model Database in search of attack templates (and corresponding arguments) that can connect attack trees. Such functions may help administrators locate previously undiscovered attacks. In addition, analytical functions may search the Attack Model Database for attack templates that accept the roots of constructed attack trees as subgoals, thereby enabling attack prediction.

VI. Conclusions and Future Work

Internet attack models must capture aggregate behavior to provide a realistic view of system exploits. Parametric attack trees offer a flexible framework for expressing exploits and multi-stage attacks. The attack visualization system described in this paper combines parametric attack trees with a system specification language to support vulnerability assessment and attack visualization. Moreover, remote attack notification services communicating with a centralized monitor allow network administrators and ISSOs to observe and respond to intrusions in real-time. Advanced functions within the monitor assist in attack prediction and can help network administrators and ISSOs identify and isolate as of yet undiscovered intrusions.

References

[1] Aslam, T., Krsul, I. and Spafford, E. Use of a Taxonomy of Security Faults, *Proceedings of the 19th NIST-NCSC National Information Systems Security Conference*, pp. 551 – 560, 1996.

[2] Feiertag, R., Kahn, C., Porras, P., Schnackenberg, D., Staniford-Chen, S. and Tung, B., Common Intrusion Specification Language (CISL), Technical Report, CIDF Working Group, 1998.

[3] Frincke, D., Tobin, D., McConnell, J., Marconi, J. and Polla, D. A Framework for Cooperative Intrusion Detection, *Proceedings of the 21st NIST-NCSC National Information Systems Security Conference*, pp. 361-373, 1998.

[4] Heberlein, L., Dias, G., Levitt, K., Mukherjee, B., Wood, J. and Wolber, D. A Network Security Monitor, *Proceedings of the IEEE Symposium on Security and* Privacy, pp. 296-394, 1990.

[5] Howard J. "An Analysis of Security Incidents on the Internet, 1989-1995," PhD Dissertation, Department of Engineering and Public Policy, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1997.

[6] Javitz, H., Valdes, A. The SRI IDES Statistical Anomaly Detector. *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 316-326, 1991.

[7] Kumar, S. "Classification and Detection of Computer Intrusions," PhD Dissertation, Department of Computer Science, Purdue University, West Lafayette, Indiana, 1995.

[8] Lackey, R. Penetration of computer systems: An overview. *Honeywell Computing Journal*, **8(2)**, pp. 81-85, 1974.

[9] Landwehr, C., Bull, A. McDermott, J. and Choi, W. A taxonomy of computer program security flaws. *ACM Computing Surveys*, **26(3)**, pp. 211-254, 1994.

[10] Lindqvist, U. and Jonsson, E. "How to Systematically Classify Computer Security Intrusions," *Proceedings of the IEEE Symposium on Security and Privacy*, pp. 154-163, 1997.

[11] http://cve.mitre.org, "Common Vulnerabilities and Exposures." The MITRE Corporation, Bedford Massachusetts, 2001.

[12] http://icat.nist.gov, "ICAT Metabase." The National Institute of Standards and Technology, Gaithersburg, Maryland, 2001.

[13] Porras, P. and Neumann, P., EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances, *Proceedings of the 20th NIST-NCSC National Information Systems Security Conference*, pp. 353-365, 1997.

[14] Schneier, B., "Attack Trees," *Secrets and Lies*. pp. 318-333, John Wiley and Sons, New York, 2000.

[15] Snapp, S., Brentano, J., Dias, G., Goan, T., Heberlein, L., Ho, C., Levitt , Mukherjee, B., Smaha, S., Grance, T., Teal, D., and Mansur, D. DIDS (Distributed Intrusion Detection System) -- Motivation, Architecture, and an Early Prototype, *Proceedings of the 14th NIST-NCSC National Computer Security Conference*, pp. 167-176, 1991.

[16] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., Levitt, K., Wee, C., Yip, R. and Zerkle, D. GrIDS – A Graph-Based Intrusion Detection System for Large Networks. *Proceedings of the 19th NIST-NCSC National Information Systems Security Conference*, pp. 361-370, 1996.

[17] http://www.sans.org/topten.html "How to Eliminate the Ten Most Critical Internet Security Threats: The Experts' Consensus, Version 1.32." SANS Institute, 2001.

[18] Vert, G., D. Frincke and McConnell, J. A Visual Mathematical Model for Intrusion Detection, *Proceedings of the 21st NIST-NCSC National Information Systems Security Conference*, pp. 329-337, 1998.