A Systematic Approach to Multi-Stage Network Attack Analysis

Jerald Dawkins
University of Tulsa
jerald-dawkins@utulsa.edu

John Hale
University of Tulsa
john-hale@utulsa.edu

Abstract

Network security analysis must coordinate diverse sources of information to support effective security models. The modeling process must capture security-relevant information about targets and attackers. By capturing the trust relationships, vulnerabilities, and attacker capabilities, a security analyst can define and characterize complex, multi-stage attacks. Along with conducting systematic analyses on multi-stage attacks, the opportunity also exists to facilitate large scale detection and visualization of security events by embedding modeling and analytical components within a more expansive security framework. This paper presents a formalism and methodology for multi-stage network attack analysis. Applications to network security management, including a network vulnerability analyzer prototype, are also described.

1. Introduction

While information system and network security professionals struggle to keep pace with constantly evolving threats, the hacker's arsenal now includes technology that conventional security tools and services cannot cope with. Blended and multi-stage attacks use stealth and intelligence to strategically maneuver and compromise a target, eluding detection and penetrating the most sophisticated defenses. Even mission critical systems and networks arrayed with state of the art security solutions are vulnerable to this new form of threat.

The challenge is even greater when security services are applied in an ad hoc manner, leaving dangerous gaps in an enterprise's defense architecture. Moreover, intrusion detection, audit and logging systems often provide sensory feedback data that cannot be effectively analyzed due to the data's sheer volume and the insular nature of the systems. These failings will continue to be exploited by hackers using increasingly sophisticated penetration strategies and tools. The only way to neutralize this threat is to develop an integrated suite of analytical tools that can reveal weaknesses in enterprise security architectures, correlate incidents, and that can model, visualize and predict multi-stage attacks.

This paper describes a comprehensive framework for network attack modeling and analysis. The framework is comprised of modeling substrates for network elements, system vulnerabilities and attacker capabilities. Attack chaining is presented as a technique that engages the models in the framework to support predictive analysis of network attacks. Applications of the framework are discussed, including a prototype that implements the modeling substrate in XML supporting attack tree generation and correlative analysis of network vulnerabilities.

2. Network Attack Modeling

Network attack modeling abstracts critical network system information in a manner conducive to security analysis. This entails modeling network elements and attacker capabilities, and subjecting the resulting models to a vulnerability analysis process.

2.1. Network Modeling

Network models for security analysis incorporate host properties, physical and logical topologies, and logical relationships [3]. *Host* objects contain properties that describe network entities, including operating system, network interfaces, applications and other elements.

Physical and *logical* topologies model network communication pathways. Hosts can be grouped into *zones*, expressing physical and logical separation. Zones are joined by *boundaries*, which restrict information flow between zones. Boundaries are manifested, for example, by routers, firewalls, or proxies. *Logical relationships* overlay network topologies, describing special communication and trust patterns between network entities.

Proceedings of the Second IEEE International Information Assurance Workshop (IWIA'04)
0-7695-2117-7/04 $ 20.00 © 2004 IEEE

Consider the example network shown in Figure 1. The network consists of four *zones* creating a logical separation between external, demilitarized, client and production networks. The *external zone* encompasses all Internet communication to and from the target network. The *demilitarized zone* includes a web application server. As shown, two firewalls define a perimeter between the Internet and local production and client networks.

The *production zone* consists of a single machine offering database services to the web application server. A *client zone* provides access points for non-production users, e.g. user workstations.

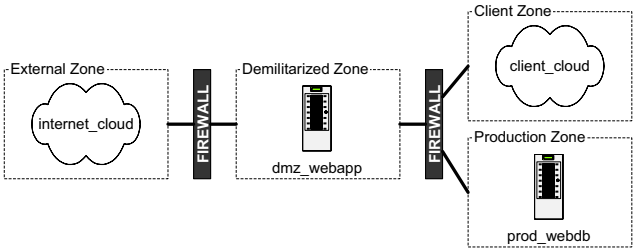

Figure 1: Example network configuration.

In this example, we adopt a simple network object model to succinctly represent network elements for security analysis. In this model, a network element is a tuple that consists of a unique identifier followed by a property in the form:

<object uid>.<property>

For example, the elements of a unique *Windows NT4* host named *dmz_webapp* running *IIS* and *WS–FTP* services can be expressed as {*dmz_webapp.nt4*, *dmz_webapp.iis*, *dmz_webapp.ws–ftp*}.

2.2. Capability Modeling

Capability models supplement network models to express attacker skill and resources. As such, they represent the state of an attacker bound to elements of the network model. For instance, a network model might represent a Windows 2000 machine, while a capability might describe an attacker's access rights to that host.

Following the network modeling example, attacker capabilities are expressed as triples consisting of an attacker identifier, the object identifier on which the capability resides, and a property:

<atkr uid>.<object uid>.<property>

For example, *atkr*'s ability to communicate with host *dmz_webapp* is expressed as {*atkr.dmz_webapp.comm*}.

	Capability	Examples
Network	location	`internet_cloud`
	operation	`arpspoof,DoS`
Host	privilege	`root,user,anon`
	access	`physical,app`
	operation	`DoS,exec,write`

Table 1. Common capabilities.

2.3. Vulnerability Modeling

Vulnerability modeling leverages network models and attacker capabilities to articulate *vulnerabilities*, *exposures* and *exploits*.

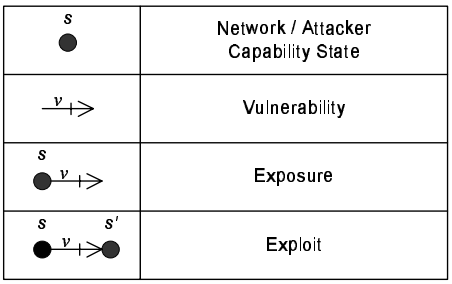

Figure 2: Diagrammatic vulnerability notation.

A *vulnerability* refers to any characteristic in a network that is a legitimate security concern.

Definition 2.1 *A* vulnerability *is a tuple* $v = (f, \mathcal{P})$*, where* f *is a state transition function such that* $f : \mathcal{S} \rightarrow \mathcal{S}$ *defines postconditions, and* $\mathcal{P} \subseteq \mathcal{S}$ *is a set of network and attacker states satisfying preconditions.*

A vulnerability contains a set $\mathcal{P} \subseteq \mathcal{S}$, where $\mathcal{P}$ defines the required network state for the vulnerability to exist, also known as preconditions. Preconditions are visually depicted by a hatch mark toward the middle of the vulnerability line in Figure 2. The postcondition function f expresses a transition from one network state to another, and is graphically represented by an arrow. The set P enables the vulnerability to express numerous network configurations. This is necessary because, a vulnerability is general in nature and does not describe any particular system.

Exposures describe specific vulnerabilities as are manifested in network elements.

Definition 2.2 *An* exposure *is a tuple* $i = (v, S, P)$*, where* v *is a vulnerability,* $S \in \mathcal{S}$ *is a network state, and* $P \in \mathcal{P}_v | \mathcal{P}_v \subset S$ *satisfies the preconditions of* v*.* $\mathcal{P}_v$ *is the collection* $\mathcal{P}$ *defined for* v*.*

An exposure i binds a given vulnerability v to a given network state S, shown in Figure 2. For some $P \in \mathcal{P}_v$, the preconditions are satisfied for that particular vulnerability. An exposure is therefore a vulnerability manifestation capable of being exploited given the current network state.

An *exploit* is the exercise of an attack on a particular exposure.

Definition 2.3 *An* exploit *is a tuple* $e = (i, S')$*, where* i *refers to a specific exposure,* $S' \in \mathcal{S}$ *is the network state defined by* $S' = f_{v_i}(S_{v_i})$*, and* f_{v_i} *is the vulnerability function associated with* v_i*.*

In this setting, an exploit is an exposure which has been attacked, resulting in an altered set of network elements S'. Thus, it triggers the application of a postcondition function to the network state S producing an augmented network state $f_{v_i}(S_{v_i}) = S'$ (Figure 2).

Two vulnerabilities are described in our example. The first, v_0, is an *IIS Directory Traversal* vulnerability that allows a malicious attacker to obtain shell access on the targeted machine. The second, v_1, is an *NT4 Privilege Escalation* vulnerability that escalates a user's access level to administrator privileges.

$v_0 =$ ($\mathcal{P}$=*{{atkr.dmz_webapp.comm,dmz_webapp.iis, dmz_webapp.win2k},{dmz_webapp.nt4, atkr.dmz_webapp.comm, dmz_webapp.iis}}*, $f(S)$=$S-${*dmz_webapp.iis*} $\cup${*atkr.dmz_webapp.usr*})

$v_1 =$ ($\mathcal{P}$=*{{atkr.dmz_webapp.usr, dmz_webapp.nt4}}*, $f(S)$=$S\cup${*atkr.dmz_webapp.root*})

The preconditions and postconditions expressed in the vulnerability models capture the functional nature of exploits over exposures. By revealing their anatomy, this formal characterization of vulnerabilities supports the systematic analysis of multi-stage attacks.

3. Attack Chaining and Analysis

The network modeling framework captures the behavior and functionality of vulnerabilities that target network elements. However, vulnerability interactions must be considered to ensure a thorough network security analysis. Vulnerability interactions can produce sequences of events, called *attack chains*.

Attack chains are informally modeled through a manual process by *red teams* [24]. However, manual construction is a tedious, error-prone and impractical process when large numbers of exposures exist [22].

Attack chaining is an iterative process that supports automated vulnerability analysis by enumerating vulnerability interactions. The process first evaluates a set of vulnerabilities to determine if preconditions are satisfied for the network state at iteration t (step 1 of Figure 3). If preconditions hold, an exposure is created for each vulnerability (step 2). Postcondition evaluation (step 3) assesses each vulnerability's postcondition function to create an augmented network state at $t+1$ (step 4). For each newly produced network state, the process continues by satisfying new vulnerabilities.

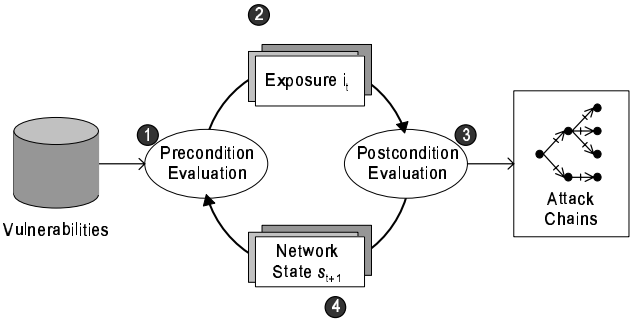

Figure 3: Attack chaining process overview.

3.1. Precondition Evaluation

Vulnerability analysis consists of the evaluation of preconditions. If preconditions hold for some network state, an exposure is created. For example, upon an initial analysis, an exposure exists on *dmz_webapp* for initial network state S_0.

$S_0 =$ {*atkr.dmz_webapp.comm, dmz_webapp.nt4, dmz_webapp.iis, dmz_webapp.ws–ftp*}

$i_{v_0} =$ {v_0, S_0, P=*{atkr.dmz_webapp.comm, dmz_webapp.nt4, dmz_webapp.iis}*}

3.2. Attack Chains

A state change occurs from network state S_0 to some altered network state S_1 upon exploiting an exposure. In this scenario, vulnerability v_0 is exploited resulting in the attacker obtaining user access on *dmz_webapp* and in the interruption of the *IIS* service process.

$e_{i_{v_0}} =$ {i_{v_0}, S_1 =*{atkr.dmz_webapp.comm, dmz_webapp.nt4, dmz_webapp.ws–ftp, atkr.dmz_webapp.usr}*}

Here, the network state S_1 enables v_1, previously with unsatisfied preconditions (the attacker needed user access to *dmz_webapp*). By exploiting v_1 the user obtains root access on *dmz_webapp* (network states S_2). This sequence of events forms an *attack chain*.

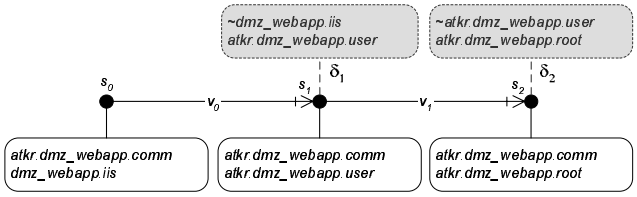

Figure 4: Attack chaining example.

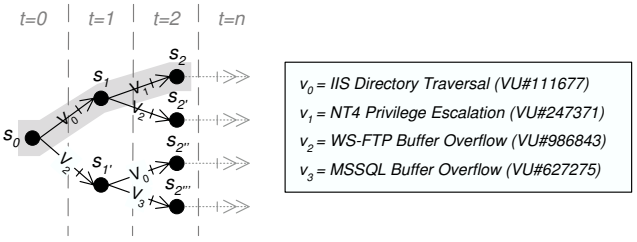

Figure 5: Attack chaining tree.

3.3. Attack Chaining Algorithm

Attack chaining is a nondeterministic application of exposures in combination to enumerate the universe of network states used to identify attack chains. The result is a tree where the root node is an initial network state. The tree consists of network states (nodes) and vulnerability exploits (edges) (Figure 5). For example, the attack chain in Figure 4 is highlighted in the attack chaining tree of Figure 5.

The attack chaining process described here engages depth-limited enumeration [17]. In attack chaining, the branching factor is a function of the number of vulnerabilities and the size of the network state space. In general, the branching factor is $O(\frac{\upsilon\lambda!}{(\lambda-d)!})$, where υ is the number of vulnerabilities, λ is the size of the universal set of network states, and d is the degree of freedom of the vulnerability model. While the branching factor complexity is potentially enormous, in practice it tends to be considerably less, due to unsatisfiable preconditions.

On the other hand, the depth of the search space is much more daunting. Since depth-limited enumeration balances storage requirements due to arbitrarily large branching factors and timing requirements for processing trees with potentially infinite depth, it is an ideal candidate for attack chaining. The attack chaining algorithm is shown in Figure 6.

The attack chaining algorithm inputs a network state S, which defines the current network state. The $depth$ argument is the current depth of the search, initially 0. The maximum depth to which the algorithm runs is defined by $max\text{-}depth$. In step 1, the algorithm checks if the maximum depth has been reached. When the maximum depth is reached, the recursion cease. Step 2 evaluates a set of exposures based upon the current network state S. Finally, for each instance $i \in I$ a recursive call is made. Notice that $f_i(s)$ evaluates

$\mathcal{V} \equiv$ The set of all vulnerabilities
$\mathcal{S} \equiv$ The set of all network states

procedure $attack\text{-}chain(S,\ depth,\ max\text{-}depth)$
input : $S \in \mathcal{S}$, the current network state
input : $depth$, the current depth of the search
input : $max\text{-}depth$, the maximum search depth
begin
1 **if** $depth = max\text{-}depth$ **then**
  $\llcorner\ return$
2 $I \leftarrow$ evaluate all vulnerabilities $v \in \mathcal{V}$ with S
3 **foreach** $i \in I$ **do**
4 $attack\text{-}chain($
  $f_i(S),$
  $(depth + 1),$
  $max\text{-}depth\ )$
end

Figure 6: Attack chaining algorithm.

the postconditions of an exposure i.

3.4. Network State Management

As new network states are created from exploits, the size of the attack chaining tree grows exponentially. Network state management deals with the complexity of network states by using network state differentials. For instance, at time t and network state S_t, initially the starting state $t = 0$, multiple instances may be exploited introducing new exposures at time $t + 1$. Network states are stored as differentials to their parents, where the root contains the network state in its entirety and children contain the changes, δ_t (see Figure 4). In essence, a network state S_t inherits the network state from S_{t-1} where the differentials are defined in δ_t. In this manner, multiple states are represented with minimal overhead. A joining function is defined by $\sqcup : (\mathcal{D} \times \mathcal{S}) \rightarrow \mathcal{S}$ where $\mathcal{D}$ is the set of network state changes and $\mathcal{S}$ is the set of network states. At any time, reconstruction of the entire network state S at time t can be achieved through the repeated application of $\sqcup$.

$$S_t = \delta_t \sqcup (\ldots (\delta_2 \sqcup (\delta_1 \sqcup S_0)) \ldots)$$

For example, consider the attack chain in Figure 4. The network state S_0 contains the initial network conditions. When the attacker exploits v_0 a new network state emerges creating a δ_1 — the network differential at time 1. Similarly, δ_2 is produced at time 2. Reconstruction of the network state S_t at any time t may be done by recursive calls to the joining function.

$$
\begin{aligned}
S_2 &= \delta_2 \sqcup (\delta_1 \sqcup S_0) \\
&= \{atkr.dmz\_webapp.root,\ \tilde{}\ dmz\_webapp.usr\} \sqcup \\
&\quad (\{atkr.dmz\_webapp.usr,\ \tilde{}\ dmz\_webapp.iis\} \sqcup \\
&\quad \{atkr.dmz\_webapp.comm, dmz\_webapp.iis, \ldots\}) \\
&= \{atkr.dmz\_webapp.comm,\ dmz\_webapp.root, \ldots\}
\end{aligned}
$$

This example demonstrates the reconstruction of the network state S_2 given the attack chain in Figure 4. At S_2 the *atkr* has gained *root* level access on the *dmz_webapp* machine.

3.5. Heuristics

Due to the complexity of both the vulnerability and network models, an exhaustive examination of preconditions is inefficient and cost-prohibitive. This complexity may be reduced by implementing network state evaluation heuristics.

In [3], Campbell cites two heuristics in precondition evaluation: reachability and weighted precondition predicates. Reachability reduces the size of the search space by limiting the precondition test to those with which the attacker can reach, or communicate. For instance, an attacker located outside the perimeter firewall will be unable to communicate with those protected by the firewall.

A weighted precondition predicate assigns a weighted value from 0.0 to 1.0 as the degree to which preconditions are satisfied. For example, a Linux box will never satisfy a Windows precondition and is thus assigned a value of 0.0. Whereas, a vulnerability which is not currently active, but might be enabled if a state change were to occur, could be assigned, for example, 0.7.

3.6. Attack Trees

Attack chains provide an automated means of attack graph construction, identifying the universe of potential exploits and their subsequent behaviors. While the information obtained is beneficial to security administrators, the power of attack chains lies in its application to sophisticated analysis techniques.

Attack chaining identifies a series of possible attack scenarios to a defined depth. In conducting such an analysis, a security administrator can identify a network state from which to extend analysis. For example, a network state may be identified that allows an attacker to obtain root privileges on a critical system. Upon further analysis, the system administrator can identify attack scenarios in which this condition holds. Goal-inducing attack chains (GACs) are attack scenarios, possibly of varying depths, in which some security-relevant properties hold. They can be used to construct attack trees to understand network vulnerability from an attacker's goal-oriented view.

A *top-level goal* is a system state identified by the attack chaining process from which to induce extended analysis. Each scenario in which the top-level goal exists is considered a *cut set* for the top-level goal. Thus, a cut set is a collection of exposures that, once exploited, induces the top-level goal.

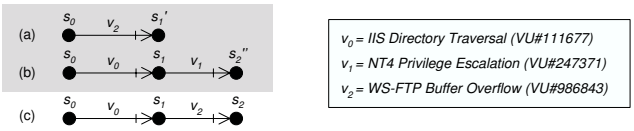

Figure 7: Goal-induced attack chains.

Figure 7 depicts attack chains for obtaining *root* access on target *dmz_webapp*. The three GACs reflect a series of unique attack scenarios. However, while attack chaining enumerates all possible combinations, some scenarios are superfluous. For example, GAC c identifies an attack scenario involving v_0 and v_2. However, since exploiting v_2 alone produces the desired failure, v_1 is irrelevant and the system will be compromised whether v_1 is exploited or not. A *minimum cut set* (i.e., a minimum GAC) is a cut set such that if any basic event is removed from the set the top-level goal will not occur, i.e. it is the smallest combination of component failures, which if they all occur will cause the top-level goal to occur [1]. Once the minimum cut sets are evaluated, construction and quantitative analysis of the attack tree can be carried out.

Attack trees can be constructed by combining minimum GACs. An attack tree, as constructed by combining minimum GACs, has the common goal identified by the GACs as its root node. The children of the root node are the exploits identified within the minimum GACs. An attack tree shows the progression of exploits to the top-level goal (Figure 8). For example, the attack tree in Figure 8 was constructed based upon the minimum GACs identified in Figure 7.

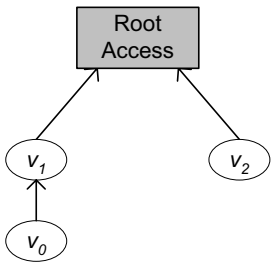

Figure 8: Attack tree.

Quantitative analysis can be carried out on attack trees to provide reliability and security risk measures of networked systems. Quantitative analysis is used to determine the reliability and performance of a system in terms of its in-

dividual constituents [1]. Attack tree analysis provides a framework in which to quantify specific system compromises. By quantifying cut sets, a security administrator can identify likely attack scenarios and appropriate countermeasures. Additionally, security improvements can be measured to provide a cost-benefit analysis.

In general, the cut set expression for a top-level goal G can be written as:

$$G = E_0 \vee E_1 \vee \ldots \vee E_n$$

where $E_i, i = 0, \ldots, n$, are the cut sets. Each cut set consists of a combination of exploits. A component cut set can be expressed as:

$$E_i = e_0 \wedge e_1 \wedge \ldots \wedge e_j, i = 0, \ldots, n$$

Where $e_0, \ldots, e_j$ is a goal-induced attack chain of size j that enables the top-level goal G.

For example, the GAC identified in Figure 7 can be used to create the attack tree in Figure 8. The GACs identify a series of attack scenarios, which yield root level access on a specified machine. The complete expression of the top-level goal *Root Access* is:

$$Root\ Access = \{e_0 \wedge e_3\} \vee \{e_1\} \vee \{e_2\}$$

The following probability functions can be applied to cut sets, to obtain a probability for each scenario to occur.

Multiplication: $P(E_i) = \prod_{k=0}^{j} P(e_k),\ i = 0, \ldots, n$

This calculation can be used to determine the most likely path an attacker can take to accomplish a given top-level goal. The probability of compromising the top-level goal G depends on each attack scenario and is defined by the equation:

Addition: $P(G) = \sum_{i=0}^{n} P(E_i)$

In the example, likelihood metrics are applied, obtaining: $P(v_0) = .15$, $P(v_1) = .65$, and $P(v_2) = .35$. ¿From this, a likelihood rating for the top-level goal is a simple application of the multiplication and addition laws of probability:

$$P(atkr.dmz\_webapp.root) = (.15 \times .65) + .35 = .4475$$

This approach results in a likelihood .4475 that the attacker will obtain root access on *dmz_webapp*. Quantitative analysis provides a perspective into obtaining a top-level goal and can be used to assess the effectiveness of countermeasures.

4. Network Security Analysis

Comprehensive network security analysis must perform a number of key functions. It must coordinate diverse sources of information; derive fully-integrated vulnerability, network, and attacker capability models; conduct vulnerability and multi-stage attack analysis; support large scale visualization and intelligent response. This section describes a network security analyzer based on attack chaining that embodies these principles.

A prototype vulnerability and multi-stage attack analyzer has been developed which leverages the power and flexibility of Extensible Markup Language (XML) [4] for attack modeling. The prototype derives its functionality from the interaction of three distinct processes: modeling, correlation and application (Figure 9). The application phase can include Attack Tree Analysis (ATA) and Attack Notification Service (ANS) components supporting preventive and mitigative security management processes.

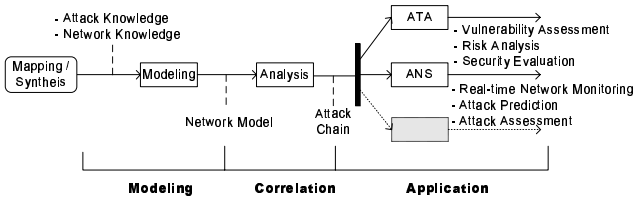

Figure 9: Network security analyzer.

4.1. Modeling

The modeling framework provides a foundation for expressing vulnerabilities, networks and attacker capabilities. These models provide a concise representation of real-world elements. The network modeling process consisting of two phases, *mapping* and *synthesis* [3]. Two common mapping techniques include *network-based mapping* and *host-based mapping*. Network-based mapping encompasses traditional remote scanning techniques. In contrast, host-based mapping tools reside locally on each host, requiring privileged access to system configuration details.

Synthesis takes as input the data generated from these mapping techniques. Synthesis engages a classification analysis that relates mapped data to the network model. In addition, topological and application-level relationships are incorporated to derive an abstract characterization of the network.

The prototype uses XML to describe the network, vulnerability and capability models. In the example below, ca-

pabilities are identified by triples and network description by tuples.

```
<!-- CAPABILITIES -->
<element>attacker.internet_cloud.location</element>
<element>attacker.dmz_webapp.com</element>
<element>dmz_webapp.prod_webdb.com</element>

<!-- NETWORK DESCRIPTION -->
<element>dmz_webapp.internet_cloud</element>
<element>dmz_webapp.ws-ftp</element>
<element>dmz_webapp.iis</element>
<element>dmz_webapp.nt4</element>
<element>prod_webdb.mssql</element>
<element>prod_webdb.nt4</element>
```

Figure 10: Network model.

Vulnerability modeling uses two XML tags: *precondition* and *postcondition*. Preconditions describe vulnerability requirements in disjunctive normal form, and are tagged by *condition*. Conditional conjunctions are grouped by the *requires* tags. The *Postcondition* tag is used to describe network state changes. These changes come in the form of additions and deletions from a network state, *put* and *del* respectively.

Network elements are bound with the use of a '?' variable. Once the variable is bound, it is substituted throughout the XML code. In addition, a wildcard '*' may be used for flexibility. In the example below, the wildcard is used to add to the network state all communication paths, denoted by *com*, from the *attacker* to those of the bound variable '?', i.e. the *attacker* can now communicate to everyone that '?' could communicate.

```
<vulnerability name="NT4 Privilege Escalation" probability="0.15">
    <precondition>
        <requires>
            <condition>attacker.?.user</condition>
            <condition>?.nt4</condition>
        </requires>
    </precondition>

    <postcondition>
        <put>attacker.?.root</put>
        <put>attacker.?.location</put>
        <put>attacker.(?.*.com).com</put>
        <del>attacker.?.user</del>
    </postcondition>
</vulnerability>
```

Figure 11: Vulnerability model.

4.2. Correlation

The correlation phase leverages the XML models via the attack chaining algorithm described in the previous section. Correlation yields complete attack chains — enumeration of all attack scenarios to a given depth.

This phase provides '?' variable binding to the XML vulnerability type definition model. Once a variable is bound, an exposure is created for that particular vulnerability. For example, given the XML data described earlier, the correlation phase would bind dmz_webapp to the variable '?' identified in the vulnerability definition. No other attributes satisfy the given vulnerability precondition. Once the '?' is bound, the vulnerability instance is maintained in a data store. The phase proceeds by continued discovery of vulnerabilities through the application of stored exposures' postcondition functions.

4.3. Application Prototype

The correlation phase supports several applications. The information generated during the modeling and correlation phase can be used in network model visualization, intrusion detection systems integration, or more sophisticated analysis techniques.

The prototype renders a two-dimensional representation of both attack chains and attack trees. The application is controlled by a simple Swing GUI component (Figure 12). The application controller allows a user to load an XML network object model, generate attack chains and attack trees, and identify potential top-level attacker goals.

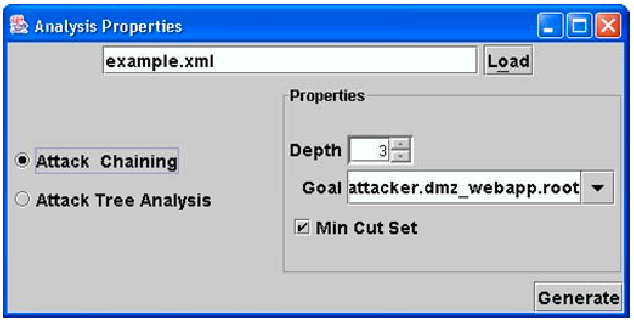

Figure 12: Attack chaining GUI.

An example run of the example network described earlier is presented here. The initial correlation phase identified the universe of network states for the given scenario to a depth of three. The enumeration reflects attack scenarios based on the models described earlier. Network states are represented as circular nodes while vulnerabilities are transitions labelled by their respective name.

Upon enumeration of the attack chains, an administrator may identify a particular goal of interest. This goal may reflect a particular condition or group of conditions that effect the operation of an organization or critical resource. In the example, the condition consisted of an *atkr* having *root* access on the *dmz_webapp* machine (*atkr.dmz_webapp.root*). Of the attack chains enumerated, a subset of those participated in such a compromise. These nodes are colored to reflect their level of participation in achieving the goal (Figure 13).

Minimum goal-induced attack chains may be identified by selecting the appropriate box. Analysis is conducted in

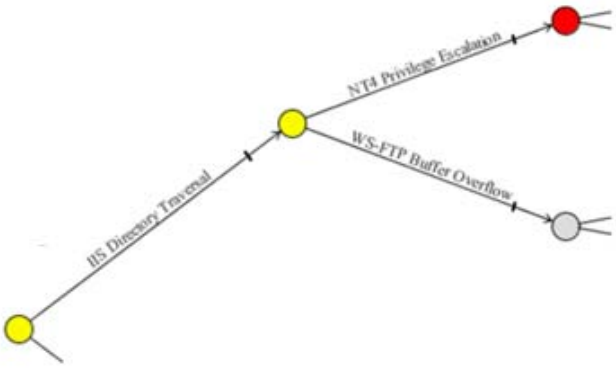

Figure 13: GAC visualization.

which all superfluous attack chains are removed. In this example the *IIS Directory Traversal → WS-FTP Buffer Overflow* was removed from the list of GACs (as shown in Figure 13).

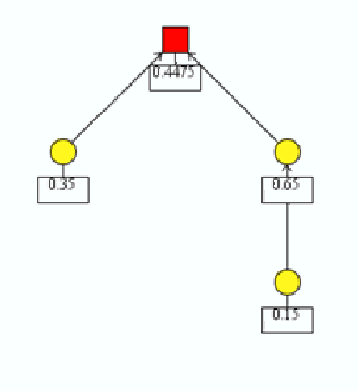

Figure 14: Attack tree analysis.

Once the minimum goal-induced attack chains are identified, quantitative analysis can be applied. The probability value expressed here relates to the potential of a given vulnerability being exploited. These vulnerability metrics may be applied for attack tree analysis, yielding a probability for a user-defined top-level goal state. A sample run is shown in Figure 14. The analysis reflects the likelihood of an *attacker* having *root* access on the *dmz_webapp* machine (*attacker.dmz_webapp.root*). In this analysis, the likelihood is shown to be .4475 that the attacker will obtain root access on *dmz_webapp*.

5. Related Work

Vulnerability analysis performs rigorous examinations to identify system vulnerabilities and other weaknesses that might allow security violations. This analysis assists a security administrator in identifying security threats and countermeasures. Research projects and commercial tools exist to eliminate vulnerabilities and prevent exploitation [5, 6, 9, 8, 11, 16, 22].

Vulnerability analysis techniques are designed for the inventory and inspection of vulnerabilities. Traditional techniques provide a detailed examination and testing of potential security flaws of a system or product. Typically, systems are cross referenced with repositories that classify and categorize of all publicly known vulnerabilities [13, 14, 21]. Some tools have been developed to assist administrators in ranking these flaws based on risk and remediation costs [2, 10, 11].

A number of security utilities automate these tasks [6, 9, 16, 18]. Some operate by scanning the target network for information such as the type and version of operating system, as well as network services configured for each host [9, 18]. This data is compared to predefined vulnerability signatures to predict which vulnerabilities are present on the network. However, this technique is only as accurate as the ability to remotely enumerate system information and the completeness of the signature database. Furthermore, the process is typically plagued with an abundance of false positives. One solution to this problem is to exploit vulnerabilities before reporting, to verify existence [6, 16]. A more passive approach is possible by comparing vulnerability signatures to information stored in a configuration management database or other network representation.

Vulnerability modeling improves the operation and efficiency of vulnerability analysis techniques by creating abstract representation conducive to further analysis. Recent advances in vulnerability modeling have yielded the representation of vulnerabilities in the form of graph-based models. Graph-based techniques capture steps of a successful system compromise, and include tree models [5, 19, 20, 25], petri nets [7, 12] and attack graphs [15].

An attack graph is a succinct representation of all paths through a system that ends in a state where an intruder has successfully achieved his/her goal [22]. Attack graphs provide a visual means to represent attack scenarios, given a particular goal. While these techniques support formal analysis, their weakness lies in their construction, as the process is typically accomplished manually. Unfortunately, complex network topologies spawn more vulnerabilities than any individual or team can reasonably hope to identify in this manner. Sheyner presents a scheme for the automatic generation of attack graphs in [22] utilizing model checkers. The effectiveness of the model is limited by the expressiveness of the model checker.

Attack trees, an extension of fault trees, give a formal methodology for describing the vulnerability of systems based on goal-oriented attack behavior [20]. Attack trees facilitate the expression of multi-stage attacks by modeling the behavior and effects of exploits. Attack trees assert

subgoals for achieving the goal set forth by an attack node [5, 25]. Attack nodes can be grouped into '*AND/OR*' sequences to capture conjunctive and disjunctive attack conditions, respectively. The nodes of attack trees can be assigned various attributes, such as cost or likelihood, in order to analyze a given series of attacks.

A petri net is a special type of attack graph consisting of places, transitions, arcs, and tokens [7]. Places are represented by circles and can be compared to nodes in a graph. Action or flow is modeled with tokens that move between places along transitions. McDermott presents *Attack Net* as an approach to attack modeling for penetration testing with petri nets [12]. In this model, attack steps are represented by places, which are similar to nodes in an attack tree. Transitions are used to model attacker actions and significantly extend the expressiveness of this model compared to attack trees. Petri nets are well-suited for IDS research projects because of their ability to represent network states and attacker's actions, nodes and arcs respectively [7, 10, 12, 23]. Thus, petri nets provide a comprehensive view of network conditions and actions needed for an attack.

Attack chains are similar in form to petri nets, in that each circle or node is a network state, and each transition is some attacker action. However, petri nets require a precognition of the search space — a predetermined knowledge of event sequences. Similarly, attack trees require an understanding of the relationship between attacks. Attack chaining aims at the automatic generation of such events, supporting the application of traditional fault tree analysis techniques.

Some attack analysis techniques support automatic construction of attack graphs [15]. While these may offer a better understanding of attack scenarios, they commonly lack the structural integrity of attack trees and petri nets. Attack chaining incorporates the expressiveness of attack graphs, without compromising the mathematical rigor necessary for formal analysis.

6. Conclusions

Increased sophistication of blended and coordinated attacks have emerged as disturbing trends from the evolving threats to enterprise networks. In such attacks, mitigation and prevention can hinge on understanding the objectives of the attacker and the logical structure of the attack. The framework we present relies on a modeling substrate to capture vital features of systems and attackers. Analytical techniques based on attack chaining can give a network administrator critical insight on the anatomy of a multi-stage attack as well as an enterprise-level view of system exposures, both directly supporting the network security management process.

References

[1] J. Andrews and T. Moss. *Reliability and Risk Assessment*. The American Society of Mechanical Engineers, 2002.

[2] T. Aslam, I. Krsul, and E. Spafford. Use of a taxonomy of security faults. Technical report, COAST Laboratory, 1996.

[3] C. Campbell. A stateful framework for multi-stage network attack modeling. Master's thesis, University of Tulsa, 2003.

[4] N. Chase. *XML Primer Plus*. SAMS Publishing, 2003.

[5] J. Dawkins, C. Campbell, R. Larson, K. Fitch, and T. Tidwell. Modeling network attacks: Extending the attack tree paradigm. In *Proceedings of Third Annual International Systems Security Engineering Association Conference*, 2002.

[6] R. Deraison. Nessus. http://www.nessus.org.

[7] G. Helmer. Software fault tree and colored petri net based specification, design and implementation of agent-based intrusion detection systems. http://citeseer.jk.nec.com/helmer01software.html.

[8] J. Howard. *An Analysis of Security Incidents on the Internet 1989-1995*. PhD thesis, Carnegie Mellon University, 1997.

[9] Internet Security Systems, Inc. Internet Scanner. http://www.iss.net.

[10] S. Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, August 1995.

[11] U. Lindqivst and E. Jonsson. How to systematically classify computer security. pages 154–163. Proceedings of the IEEE Symposium on Security and Privacy, 1997.

[12] J. McDermott. Attack net penetration testing. pages 15–22. ACM SIGSAC, The 2000 New Security Paradigms Workshop, 2000.

[13] Mitre. Common vulnerabilities and exposures. http://cve.mitre.org.

[14] T. National Institute of Standards and Technology. ICAT metabase. http://icat.nist.gov.

[15] C. Phillips and L. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 Workshop on New Security Paradigms*, pages 71–79. ACM Press, 1998.

[16] Rapid7, Inc. NeXpose. http://www.rapid7.com.

[17] S. Russell and P. Norvig. *Artificial Intelligence A Modern Approach*. Prentice Hall, 2003.

[18] SAINT Corporation. SAINT Scanning Engine. http://www.saintcorporation.com.

[19] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, pages 21–29, 1999.

[20] B. Schneier. *Secrets and Lies*, pages 318–333. John Wiley and Sons, 2000.

[21] Security Focus. Bugtraq. http://www.securityfocus.com.

[22] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2002.

[23] M. Slagell. The design and implementation of MAIDS. Master's thesis, Iowa State University, 2001.

[24] J. Somesh and W. J. Sheyner, O. Minimization and reliability analyses of attack graphs. Technical report.

[25] T. Tidwell, R. Larson, K. Fitch, and J. Hale. Modeling internet attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, June 2001.