Representation and Analysis of Coordinated Attacks

Sviatoslav Braynov and Murtuza Jadliwala
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260
{sbraynov, msj3}@cse.buffalo.edu

ABSTRACT

In this paper, we propose a formal model of coordinated attacks in which several attackers cooperate towards a common malicious goal. The model investigates both attack planning and vulnerability analysis, thereby providing a uniform approach to system and adversary modelling. In addition, the model is general enough to explain both coordinated and single attacks.

In the paper, we define the notion of coordinated-attack graph, propose an algorithm for efficient generation of coordinated-attack graphs, demonstrate how coordinated-attack can be used for vulnerability analysis, and discuss an implementation of a coordinated-attack graph.

Coordinated-attack graphs can facilitate a wide range of tasks, such as model checking, opponent modelling, intrusion response, sensor configuration, and so forth. In addition, they can be used in robotic warfare, where several intelligent software agents automatically produce and launch coordinated attacks.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification — *formal methods, model checking*

General Terms

Security

Keywords

Coordinated attack, attack graph, adversary modelling, attack plan, model checking

1. INTRODUCTION

Organized attacks aiming at disrupting critical infrastructure are usually beyond the power of a single attacker. To achieve their goal, several attackers cooperate by resource sharing, task allocation, and synchronization. Coalitions of malicious attackers may include both human and artificial agents, i.e., intelligent software agents acting on behalf of humans. A recent CERT report [13] concludes that modern attack tools are rapidly evolving and becoming more sophisticated. Unlike early attacks, launched by a single attacker to a single victim, recent attacks are better coordinated and difficult to discover. A notorious example is the October 23rd attack on DNS root servers [20].

Most literature on distributed attacks has traditionally focused on distributed denial of services (DDoS), in which an attacker breaks into several machines, or joins other attackers to simultaneously attack a target host or network. A distributed attack, however, is a very simple form of coordinated attack, where many intelligent attackers coordinate in real time. Given the present state of coordinated attacks, we could expect new and more damaging patterns of distributed attacks in the near future. Our concern is that current attack patterns have not utilized the full potential for real time coordination and cooperation.

In our previous research [5], we identified several problems with coordinated attacks:

- **Coordinated attacks could be designed to avoid detection.** Many defense technologies can be easily defeated by a sophisticated coordinated attack capable of breaking the attack pattern into many apparently innocent pieces. Think of the following example. Consider a large crowd (users, processes, hosts, threads) with few attackers hiding inside it and executing a coordinated attack plan. Due to the large crowd, it is practically impossible to discern individual attackers many of which could be performing apparently innocuous actions. Once the attack succeeds it is clear who the attackers are, but at this point it is too late to make a difference.
- **It is difficult to differentiate between decoy and actual attacks.** Consider the case where members of a malicious group launch several simultaneous attacks on a system. In order to mislead the intrusion response system, all but one of these attacks are designed to be decoy. That is, they are launched for the sole purpose of distracting the intrusion response system and consuming its resources. Decoy attacks may have different goals. They could: create many simultaneous alerts in order to mislead or confuse the IDS; waste system response time on decoy goals; or perform a DoS attack on the IDS.
- **There is a large variety of coordinated attacks.** Attackers may attack single or multiple victims. Some attacks could be automatically replicated or duplicated, thereby accumulating more power. Other attacks may be supporting, i.e.,launched in parallel with the main attack, with the aim of providing auxiliary functionality such as cover-up, back-up, trace removing, etc.

Although the paper discusses primarily insider attacks, the models presented are general enough to be applied to a wide range of scenarios including insider, outsider, and mixed attacks.

FMSE'03, October 30, 2003, Washington, DC, USA.

Since the actions of different attackers interact, we cannot specify the effects of an individual attacker's action without taking into account what actions might be performed by other attackers at the same time. It is often the case that the effect an action is modified by another action executed concurrently by another attacker. In other words, the result of a joint action, executed simultaneously by several attackers, could go beyond the sum of individual actions.

Current research on vulnerability analysis, including graph-based systems [22, 24, 17, 15, 16, 23] and correlation-attack languages [7, 21, 26], has mainly focused on serial attacks, i.e., sequences of atomic actions leading to a security breach. Although correlation analysis has been applied to analyze relationships between different ongoing attacks, little attention was paid to correlating individual actions across users in order to identify a single attack. In a coordinated attack, attackers' actions interfere with one another, making it difficult to analyze an action out of the context of other actions. Consider, for example, two legitimate users with different access rights. While the first user has access to sensitive data, and no access to an external network, the second user has external network access, and no access to sensitive data. The users could collude and launch a coordinated insider attack. They could first establish a covert channel between them, and then let the system leak sensitive information. In this case, most intrusion detection systems would detect an authorized access to sensitive data and an authorized network access, without being able to correlate them.

The simplest way to handle interactions between concurrent attackers' actions is to consider each joint action as an atomic action, i.e., to consider the group of attackers as a single attacker. Specifically, if A_i denotes the actions available to attacker i, then the joint action space is $A_1 \times A_2 \times ... \times A_n$. That is, a joint action is a combination of all individual actions taken at the same time. One may see each element of the joint action space as an atomic action, and specify its effect using existing specification languages such as STATL [21] and P-BEST [7]. The main advantage of this reduction is that vulnerability analysis can be performed using available methods and tools. Such an approach, however, has some serious disadvantages as far as expressiveness is concerned, as well as ease of representation and analytic power. First, the number of joint actions increases exponentially with the number of attackers (assuming that each attacker can execute at most one action at a time). This leads to a super-exponential blowup of specification and system models. Second, this reduction fails to account for the interaction between attacker's actions, and thereby to provide data for further correlation analysis. Third, it fails to exploit the fact that many attackers'actions may not interact at all, or interact under some conditions only.

In this paper we propose a formal model of coordinated attacks in which several attackers coordinate their actions to achieve the common goal of compromising a computer or network system. A distinctive feature of the model is its ability to account for concurrent interdependent attackers' actions. In a concurrent-action attack, each individual action can bring about the desired effect only if properly coordinated with other actions which could be concurrently or sequentially applied.

The paper is organized as follows. Section 2 introduces a formal model of concurrent systems. Section 3 defines the notion of coordinated attack plan. Section 4 discusses how several attack plans may combine into a coordinated-attack graph. The section also presents an efficient algorithm for generating attack graphs. Section 5 illustrates how coordinated-attack graphs can be used for vulnerability analysis. Finally, Section 6 discusses an implementation of a coordinated-attack graph.

2. FORMAL FRAMEWORK

In this section, we describe a model of coordinated attacks that accounts for synergy and coordination among attackers.

1. S is a finite set of system states
2. $S_0 \subseteq S$ is a set of initial sets
3. System states are truth assignments to ground atomic formulae. A state is represented as a set (or conjunction) of those ground atomic formulae that are true in the state. For example, the state in which user1 has logged and accessed file1 is represented as:

   $$logged(user1) \wedge accessed(user1, file1)$$

   This implies the closed world assumption, i.e., every atomic formula not listed in a state evaluates to FALSE.
4. A_i is the set of actions available to attacker i. The joint action space is $A = A_1 \times A_2 \times ... \times A_n$. That is, each joint action $\bar{a} = (a_1, a_2, ..., a_n), a_i \in A_i$, is a combination of individual actions performed by each of the attackers. Throughout this paper, we assume that an attacker can perform at most one action at a time. Some of the actions could be ϵ, a null or no-op action. We assume that $\epsilon \in A_i$ for $i = 1, ..., n$. In other words, some attackers could be idle at some stages of the attack.
5. $T \subseteq S \times A \times S$ is a trinary relation, the *transition relation*, which gives possible transitions between states. That is, $T(s_1, \bar{a}, s_2)$ describes a transition from state s_1 to s_2, if joint action $\bar{a}$ is taken in state s_1. In this paper, we constrain our attention to deterministic transitions. The model can easily be generalized to handle nondeterministic actions by introducing a probability distribution on the space of resulting actions. Note that the joint action $\bar{a}$ is not an atomic action, but a vector of individual actions, each defined separately. Since the effects of the concurrent actions applied to s_1 interfere, the resulting state s_2 is determined by all concurrent actions.
6. G is the attackers' goal, i.e., a first-order formula describing a compromised system state. $S_G, S_G \subseteq S$ is a set of system states in which the goal is satisfied. Note that, G could be a complex goal consisting of several concurrent goals. In this case, it could be represented as a conjunctive lists of formulas.

For the ease of notation, each action is described by a generic action schema specifying: who is performing the action, what are the action preconditions, what other actions must be performed concurrently, and what is the final effect of the action. Figure 1 describes the format of an action schema. In a schema, action preconditions specify which atomic formulae must be true in the current state of the system in order to apply the action. The postconditions specify which atomic formulae become true and which become false after the action is executed. The concurrent list specifies other actions that must be simultaneously executed or not executed for a given action to have its intended effect. To specify a concrete action, all free variables in a schema must be bound to constants. In other words, every action is a fully instantiated action schema. We assume that conjunctive lists, pre- and postconditions are consistent, i.e., jointly satisfiable. That is, there exists at least one variable assignment that satisfies them in at least one system state. For example, a concurrent action list must not require that a particular action be applied and not applied at the same time in a given state.

action <first-order predicate>
:parameters
<list-of-free-variables>
:preconditions
<conjuctive-list-of-predicates>
:concurrent <conjuctive-list-of-action-names>
:postconditions <conjuctive-list-of-predicates>

Figure 1: Action schema

An action (a fully instantiated action schema) can be executed only if it's preconditions are true. After the action execution, the system state is changed in the following way. All positive literals from postconditions are added into the state description while all negative literals are removed. For example, if executed in state

$$logged(user1) \wedge accessed(user1, file1)$$

a logging off action with a postcondition *not(logged(user1))* will change the current state to

$$accessed(user1, file1))$$

Our action representation significantly differs from representations used in attack graphs. First, it allows for free variables, thereby allowing an action schema to present a whole class of instant actions. When an action schema is instantiated, all variables must be bound to constants. Second, it allows for concurrent actions, and explicitly describes action dependence. Synchronization and coordination among attackers is captured by the concurrent action list. It specifies what actions must be executed concurrently in order to enable positive synergy, and what action combinations are prohibited in order to avoid negative synergy. That is, some actions must be taken (for positive synergy) or must not be taken (for negative synergy) in order for a given action to have its intended effect specified by postconditions. Since an attacker can perform at most one action at a time, all concurrent actions must be performed by different attackers.

Figure 2 represents two concurrent actions corresponding to a symbolic link race condition [10]. In this attack, two insiders, *user1* and *user2*, cooperate in order to gain access to *file*, which they are not authorized to access. *User1* creates symbolic link *file1* pointing to *file2*. We suppose that *user1* has access to read and write both *file1* and *file2*. After establishing the symbolic link, *user1* calls open(file1, O_RDWR). The system resolves the symbolic link and checks that access to *file2* is allowed. Meanwhile, *user2* changes the symbolic link *file1* to point to *file*. As a result, the system executes open(file, O_RDWR), granting *user1* access to *file*.

The intended semantics of an action scheme is captured by the following definition.

DEFINITION 1. *A joint action $\bar{a} = (a_1, a_2, ..., a_n), a_i \in A_i$, a combination of individual actions, is consistent in the current state s iff:*

- *The preconditions of all individual actions are jointly logically consistent in s. That is, the conjunction of all atomic formulae in precondition lists evaluates to TRUE in s.*
- *The postconditions are jointly consistent in s. That is, there are no two actions such that the first one brings about a ground atom while the second action brings about the negation of that atom. In other words, action effects do not conflict with one another.*

action open(user1,file,O_RDWR)
:parameters
user1, file, file1, file2
:preconditions
can-access(user1, file1,O_RDWR)∧
can-access(user1, file2,O_RDWR) ∧
symbolic-link(file1,file2)
:concurrent change-link(user2, file1, file)
:postconditions opened(user,file, O_RDWR)

action change-link(user2, file1, file)
:parameters
user2, file, file1
:preconditions
can-access(user2, file1,O_RDWR)
:postconditions symbolic-link(file1,file)

Figure 2: Action description

- *Each action a_j mentioned in the concurrent list of some a_i belongs to the joint action. In other words, it is executed by some other attacker j, different from i.*
- *The negation of action a_j, mentioned in the concurrent list of some a_i, does not belong to the joint action. In other words, no other action negatively interferes with a_i.*

If a joint action $\bar{a}$ is consistent in s, then it can be executed in that state. This results in a concurrent execution of several actions. The resulting state is obtained by adding all positive literals and deleting all negative literals from the postcondition lists of concurrent actions.

3. A COORDINATED ATTACK PLAN

In an adversarial environment, the best defence strategy depends on what the defendant believes about the attacker's strategy , which in turn depends on what the attacker believes about the defendant strategy, and so forth, leading to an infinite recursion of beliefs. In game theory and artificial intelligence, several methods have been proposed to represent and reason about adversaries [6]. Current research on vulnerability analysis usually represents an information system only from the perspective of the security analyst, thereby missing the other part of the equation. As Sun Tzu puts it in his treatise on the art of war [25]: "If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat."

In this section, we propose a formal model of coordinated attacks which can serve both as a semantic model of a concurrent computer (or network) system and as a planning model for a group of attackers. That is, the model incorporates the viewpoints of both the security analyst and the attackers. This is quite a natural approach, given that an attacker usually launches an attack based on a model of the system, and the system analysts evaluates a model against a set of possible attacks. In addition, the model is general enough to explain both coordinated and single attacks.

The semantics of individual actions in coordinated attacks is different from their semantics in individual attacks. In coordinated attacks, it is not individual actions that transform one state of the system into another. Rather, the state transitions are triggered by joint actions concurrently executed by several attackers.

DEFINITION 2. *An individual plan p_i for attacker i is a sequence of actions*

$$p_i = (a^i{}_0(s_0), a^i{}_2(s_1), , a^i{}_k(s_k))$$

where action $a^i{}_j(s_j)$ is executed in state s_j by attacker i.

An individual plan is deterministic in the sense that each attacker knows what action to execute at every state of the system. At first sight, this might seem a limiting assumption since attackers are usually prepared to react to unforeseen failures and contingencies. For example, attackers often employ contingency plans that allow them to recover from unexpected accidents. Such scenarios traditionally fall in the domain of contingency planning [19, 3]. Using standard methods from contingency planning, the model presented in this paper can easily be extended to include conditional actions.

We assume that the system state is accessible for attackers in the sense that they have the knowledge and the skills to differentiate between the states in the attack plan. This assumption prevents miscoordination caused by inability of an attacker to correctly determine the present state of the system. By this, we do not assume that attackers have complete knowledge of every situation they could encounter. First, attackers need to differentiate between attack-relevant states, not between all possible states. Second, in order to differentiate between relevant states, attackers need only to know or discern some distinctive features of the states.

If we combine all actions executed at a certain stage of the attack $j, j = 1, ..., k$, we obtain the attackers' joint action at that stage. It is natural to define a coordinated attack plan as a sequence of attackers' joint actions.

DEFINITION 3. *A coordinated-attack plan P for a group of attackers G is the union of coordinated individual plans of the members of the group. The coordination between attackers is specified at action level trough precondition and concurrent conditions.*

That is, a coordinated-attack plan P is a sequence of joint actions:

$$p = (a_0(s_0), a_1(s_1), , a_k(s_k))$$

where a joint action $a_j(s_j)$, taken in state s_j, is defined as the combination of all concurrent actions executed by each attacker in s_j:

$$a_j(s_j) = (a^1{}_j(s_j), a^2{}_j(s_j), ..., a^n{}_n(s_j))$$

Every coordinated-attack plan starts in the current state s_0 and finishes in some final state where the attackers's goal (or goals) is achieved, i.e., the system security is compromised. The plan specifies what action has to be taken by each agent in each state, in order to reach the goal state. At first sight it may seem counterintuitive that the attackers' goal is always satisfied in the final state. The goal of planning, however, is to plan for success. After all nobody plans for failure. Whether or not a plan will succeed depends on the validity of its assumptions.

A coordinated-attack plan can be conveniently represented as a directed and acyclic multigraph where nodes represent system states and arcs correspond to actions. All arcs leaving a node represent actions concurrently started at that node. Similarly, all arcs pointing to a node represent actions concurrently finished at that node. The intended meaning is that all arcs between two nodes must be traversed concurrently (by different attackers), in order to move the system from the first to the second state. Figure 3 shows a coordinated-attack plan for the race condition example. The attackers' goal is to open file *tf*, which they are not authorized to access. They start in the initial state s_0 and achieve their goal of opening *tf* in s_5. Note that, in order to move the system from state s_2 to s_5, *user1*'s action $a^1{}_2$ has to be concurrently applied with *user2*'s actions $a^2{}_2$, $a^2{}_3$, and $a^2{}_4$. That is, links connecting s_2 and s_5 must be traversed concurrently.

It is worth noting that a coordinated-attack graph has no loops. After all, nobody plans to take an action that has no effect. Since our plans are deterministic, there are no cycles, either. A contingency plan, however, may have cycles representing recoveries (backtracking, for example) from partial failures.

Each coordinated-attack plan is linear, since it induces a total order on the set of states. In Figure 3, the total order is represented as a chronological time line starting with the initial state s_0, followed by intermediate states and the final state s_5.

4. ATTACK GRAPH FOR COORDINATED ATTACKS

It is often the case that a group of attackers comes up with different plans for achieving their goals, and it is up to the group to choose which plan to follow. To evaluate system vulnerability, a security analyst needs a model of possible attacks. An attack model can also facilitate *model-based intrusion detection*, in which alerts are matched against the model to discover attackers' coordination patterns.

DEFINITION 4. *An **adversary's instrumental capacity**, (AS, n), is represented by:*

- *AS: The actions available to attackers.*
- *n: The maximal number of attackers participating in an attack.*

The notion of adversary's instrumental capacity covers knowledge, skill levels, and tools available to attackers. In this paper, we consider homogeneous groups of attackers, i.e., groups of equally skilled and knowledgable attackers. In other words, if an attacker can accomplish a task, so can any other member of the group. Apparently, attackers share knowledge, tools, and even teach one another time permitting. The homogeneity assumption also applies for software agents acting on behalf of human attackers. A software agent can easily replicate and launch new copies of itself.

Homogeneous attackers can benefit in many ways from cooperation. For example, in order to make an attack short, independent subtasks can be assigned to different attackers. In some cases, homogeneous attackers can cooperate to avoid detection (parallel port scanning). It is also possible that some steps of an attack may require a joint action that is beyond the power of a single attacker.

Note that the notion of instrumental capacity does not cover attackers' decision-making skills. Different groups of attackers can utilize the set of available actions AS in different ways, some groups making more efficient use of actions than others.

DEFINITION 5. *A coordinated-attack graph is the union of all coordinated-attack plans, which for a given adversary's instrumental capacity (AS, n), reach a goal state $s, s \in S_G$, when applied to the current state S_0. More formally, a coordinated-attack graph is:*

$$CAG = (V, E, \alpha, S_0, S_G, AS, n)$$

where (V, E) is a multigraph of system state transitions. Each node $v \in V, V \subseteq S$, represents a system state, and each arc $a, a \in AS$, represents an attacker's action. The action set AS is the set of all actions available to attackers, and n is the maximal number of attackers that can participate in the attack. S_0 is the current state, and S_G are the goal states. α is a labelling which associates each arc with an action in AS.

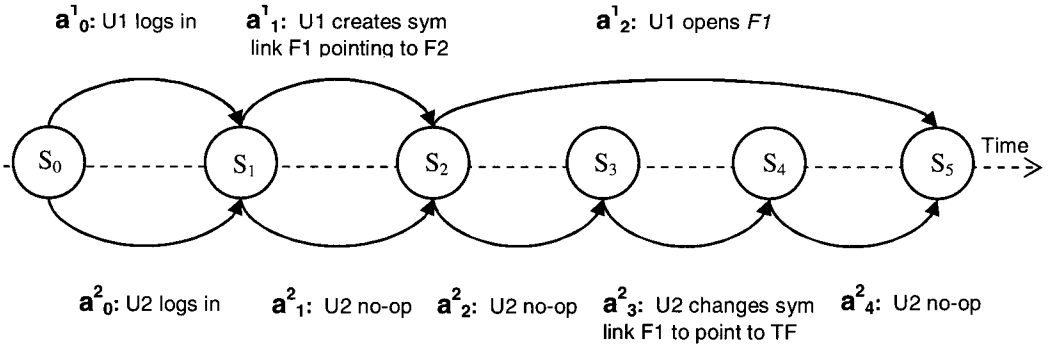

Figure 3: Coordinated-attack plan

As shown, a coordinated-attack graph is the union of all coordinated-attack plans that can reach the goal G, i.e., the plans having a final state in S_G. A coordinated-attack plan can be automatically generated by any sound and complete planning algorithm that allows for concurrent actions, such as POMP. A planning algorithm is *sound* if it generates only plans that are guaranteed to achieve the attackers' goals. A *complete* algorithm is guaranteed to generate a plan, if a successful plan exists. Obviously, a complete planning algorithm can iteratively generate all existing plans, by disabling previously generated plans.

Figure 4 is an example of a coordinated attack graph. The lower branch of the attack graph represents the race condition attack discussed above. The upper branch represents a similar attack called directory race cognition [10]. The dotted lines in Figure 4 are used to join concurrent links. That is, the two links leaving node s_0 are concurrent, whereas the two links leaving s_1 represent a decision choice in which attackers have to chose from two available attack continuations.

Coordinated-attack graphs represent a semantic model of a system and can facilitate a wide range of tasks, such as model checking, opponent modelling, intrusion response, sensor configuration, and so forth. For example, a security feature of a system can be proved using standard model-checking techniques. Current research on planning as model checking allows one to check a plan for its correctness, or to validate a property of a system written in Computation Tree Logic (CTL)[12].

There has been a lot of research on planning and concurrent planning. And yet surprisingly, we could not find an efficient implementation of a concurrent planner. POMP-based planners [2], for example, fail to handle large state-action spaces. The exponential size of the action-space state could eventually be handled by model-checking planners, such as UMOP [14] and MBP [1] which make use of Ordered Binary Decision Diagrams (OBDD). OBDD's are very compact and efficient representations of the assignments satisfying a given boolean formula. Planners using OBDD could handle state-action spaces of magnitude 10^{20} and higher [18]. Unfortunately, neither MBP nor UMOP allow for concurrent planning.

One major problem with all existing planning and model-checking systems is that they do not exploit action interdependencies to prune the state-action space. For example, a planning system would search for all possible action combinations applicable in a given state. The situation is further exacerbated, by the exponential growth of the number of states.

Concurrent actions might depend on one another in that an action execution might require the concurrent execution of another action. Such a dependence helps avoiding some futile search and increase planning efficiency. For example, if an action is not applicable in a state, neither are the actions depending on it. That is, all actions that require the concurrent execution of the given action can be discarded. Action dependencies can be detected efficiently in polynomial time, and all inapplicable actions could be discarded, thereby significantly reducing the search space.

In this paper, we propose a method for pruning the size of coordinated attack graphs. The method uses a data structure, the action-dependence graph, to explicitly represent dependencies and interaction between concurrent actions.

DEFINITION 6. *For a given adversary's instrumental capacity (AS, k), an **action-dependence graph** is a graph-based representation of the action set AS, where each action is represented by a node. A link is drawn from node a_1 to node a_2 iff a_2 belongs to the concurrent list of action a_1, i.e., the execution of a_1 requires the execution of a_2.*

The fact that the execution of action a_1 requires the concurrent execution of a_2 can be viewed as a dependence of a_1 on a_2, i.e., action a_1 cannot be executed unless a_2 is concurrently executed. Dependence is not a symmetric relation. In Figure 3, for example, action $a^1{}_2$ depends on $a^2{}_3$. In other words, *user1* cannot open *file* unless *user2* concurrently changes the symbolic link. The reverse, however, is not true. An example of an action-dependence graph is

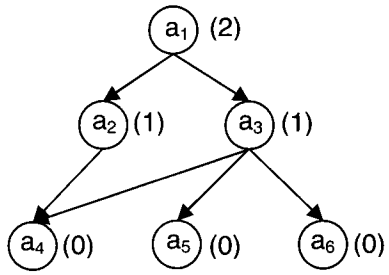

Figure 5: Action-dependence graph

shown in Figure 5. In this case, action a_1 requires the concurrent execution of actions a_2 and a_3. Action a_2 requires action a_4, and so forth.

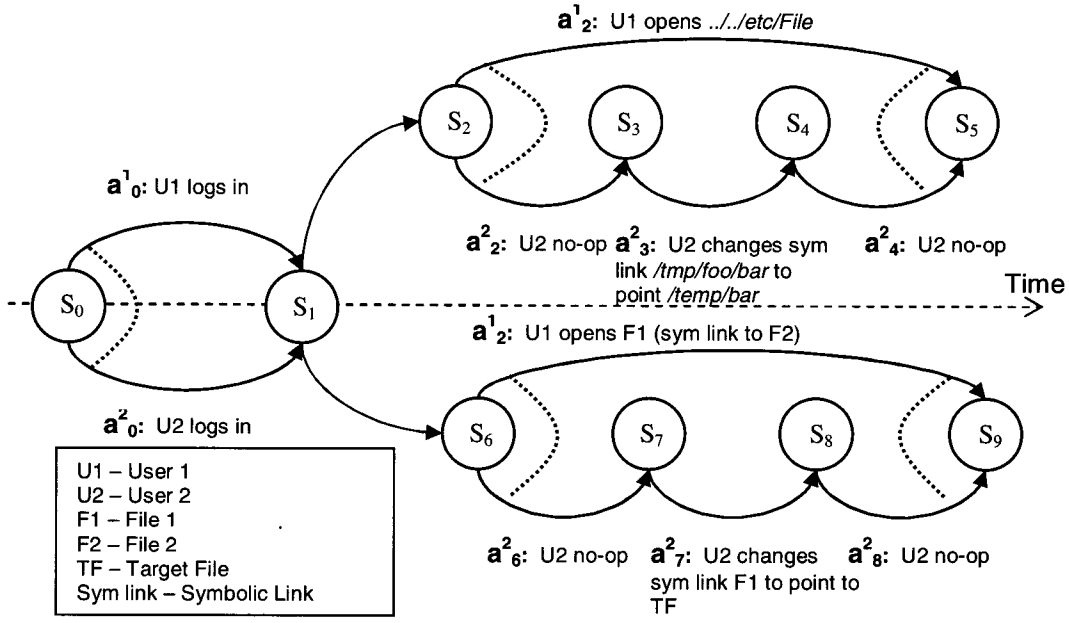

Figure 4: Coordinated-attack graph

The action-dependence graph in Figure 5 suggests that some actions are more "costly" than other actions. For example, if actions a_2 and a_4 can achieve the same goal, there is no point in using a_2. If a_2 is to be used, then it requires a_4, which by itself is sufficient for achieving the goal. This bring us to the following observation.

OBSERVATION 1. ***Minimum concurrency:*** *A set of attackers tries to achieve their goal with minimal sets of concurrent actions.*

Put formally,

DEFINITION 7. *A set of concurrent actions Z is* ***irreducible,*** *if no proper subset of Z has the same effect as Z.*

In other words, there is no point in using a large set of concurrent actions if a smaller subset can achieve the desired effect. There are several reasons for choosing a small set of concurrent actions at each stage of an attack. First, small action sets could decrease the cost of each stage, and therefore the total cost of an attack. Second, choosing more actions requires satisfying more preconditions, unnecessarily increasing the complexity of an attack. Third, small action sets can be carried out by a small number of attackers.

The action-dependence graph in Figure 5 shows that actions a_4, a_5, and a_6 are independent. That is, each of them can be executed on its own. The same cannot be said, however, for actions a_1, a_2, and a_3, all of which depend on other actions. Intuitively, actions a_1, a_2, and a_3 have different degrees of dependence. For example, action a_1 requires the execution of two other actions, while action a_2 requires only one. More formally,

DEFINITION 8. *The degree of dependence of action a is:*

$$d(a) = \begin{cases} 0 & \text{if action } a \text{ is independent,} \\ m+1 & \text{otherwise.} \end{cases}$$

where m is the maximal degree of dependence of immediate successors of a.

Definition 8 is only applicable to acyclic action-dependence graphs. To apply it to cyclic graphs, we need to transform the initial graph to an acyclic one in which every node corresponds to a maximal strongly connected components of the initial graph. The degree of dependence of a node in the original graph is then defined as the degree of dependence of the strongly connected component to which that node belongs.

In Figure 5, the degree of dependence is shown in parenthesis next to each action. An action-dependence graph and degrees of dependence can be computed in polynomial time using standard graph algorithms. In Figure 6, we present the DBCP (Dependency-Based Concurrent Planning) algorithm, which makes use of an action-dependence graph to prune the search space. The algorithm has four input variables: the set A of all joint actions inserted into the plan so far, the current subgoal $Subgoal$, the set of available actions AS, and the set of available agents n. $Subgoal$ is a list of preconditions that have not been achieved. At the beginning, it is initialized to the goal G. The algorithm works backwards. That is, it starts with the goal preconditions and tries to satisfy them by choosing appropriate actions, which in turn requires satisfying new preconditions, and so forth, until all preconditions are satisfied or cannot be satisfied, given the set of available actions AS and number of agents n. At each step DBCP chooses a minimal action set that satisfies a subgoal, thereby reducing both the cost of search and the total cost of the plan.

DBCP(A,Subgoal, AS, n)
A$\leftarrow \emptyset$
AvailableAttackers$\leftarrow n$
Satisfied$\leftarrow \emptyset$
while (Subgoal $\neq \emptyset$) and (AvailableAttackers > 0)
do
choose Precondition Q **from** $Subgoal$
choose action A_{add} **from** AS with minimal dependence that brings about Q
If A_{add} does not exist **then return failure**
Satisfied$\leftarrow$ Satisfied $\cup\, Q$
JointAction $\leftarrow A_{add}$
Subgoal$\leftarrow$ Subgoal $- Q$
AvailableAttackers $\leftarrow$ AvailableAttackers $- 1$
for each Postcondition P **in** A_{add}
if $P \notin$ Satisfied **then** Satisfied $\leftarrow$ Satisfied $\cup\, P$
for each action A_{con} on which action A_{add} depends
do
Flag $\leftarrow$ false
for each precondition P **in** A_{con}
do
if $(P \notin Q)$ and $(P \notin$ Satisfied$)$ **then**
do
$Q \leftarrow Q \cup P$
Flag $\leftarrow$ true
enddo
enddo
if (Flag) **then**
do
if (AvailableAttackers $= 0$) **then return failure**
else AvailableAttackers $\leftarrow$ AvailableAttackers $- 1$
JointAction $\leftarrow$ JointAction $\cup\, A_{con}$
enddo
enddo
$A \leftarrow A \cup$ JointAction
enddo
if Subgoal $= \emptyset$ **then return** A **else return failure**

Figure 6: Dependency Based Concurrent Planning (DBCP) algorithm

5. VULNERABILITY ANALYSIS FOR COORDINATED ATTACKS

Planning used for model checking has proved to be practical and general enough to tackle a wide range of problems [12]. Efficient planning algorithms (including OBDD-based algorithms) can generate plans automatically and deal with large problem spaces. Using planning for vulnerability analysis has several advantages:

- It provides a uniform approach to both system and adversary modelling.
- When a security property is invalidated by a planner, the planner produces a coordinated-attack plan that can be used by attackers to synchronize their actions. Moreover, a complete planner can generate all plans invalidating the security property. Knowing the attackers' plans can help a security analyst to prevent attacks and plan incident response.
- A coordinated-attack model can be used for a *model-based intrusion detection* which looks for specific coordination patterns between users' activities.
- Planning can help not only vulnerability analysis, but also robotic attacks in which several intelligent agents automatically produce and launch a coordinated attack.

To perform plan-based model checking, we use the adversary's instrumental capacity (AS, n) as a system model, LPG as a complete and sound planner [11], and a specification of the security property p written in CTL. Unlike other system models implemented as a finite state transition system, our model is parametric: it depends on the number of available attackers, n. For different n, the planner will generate different transition systems. At first sight, it seems unusual that the system model depends on an exogenous parameter, n. On the other hand, it is quite natural to evaluate a system against different attack scenarios. For example, a system which is safe for a given set of actions AS and a given number of attackers n, might not be safe for the same action set AS and $n+1$ attackers.

In our case, model-checking works as follows. For a given safety property p written in CTL, we generate the set of all states where p holds:

$$States(p) = \{s \in S, p \in L(s)\}$$

Then, we set the goal states $S_G = States(p)$. Finally, we run the planner with AS, n, S_G as input variables. A complete planner is guaranteed to generate a plan (if such a plan exists) starting at the current state S_0 and reaching one of the goal states $s, s \in S_G$. The existence of a plan invalidates the security property p, while the non-existence validates it. Note that a property is always checked for a fixed set of actions AS, and the maximal number of attackers.

6. IMPLEMENTATION

In this section , we show how the race condition attack shown in Figure 3 can be modelled and analyzed using Planning Domain Description Language, PDDL, and implemented in LPG (Local search for Planning Graphs) [11]. We have chosen the latest version PDDL 2.1 used in the Third International Planning Competition in 2002 [9]. PDDL 2.1 is a versatile description language capable of expressing temporal and numerical properties of planning domains [8].

Before choosing LPG, we studied the applicability of several planners to the coordinated attack domain. A major challenges was the representation of concurrent actions. Initial attempts to model the coordinated attack domain using non deterministic planners, such as MBP [1], failed. One of the major drawbacks of MBP is its inability to support temporal and numerical features of actions, such as action duration and the invariants that must hold over the duration. In PDDL, a durative action is represented by two atomic actions corresponding to the end points of the durative action. PDDL also allows for concurrent execution of actions, provided that the end-points of one action do not interact with the end points or with the invariants of another action.

LPG uses a compact representation of a planning graph to define a search neighborhood and to evaluate its elements using a parameterized function. The function assigns weights to different types of inconsistencies in the current plan, and dynamically evaluates them during search, using discrete Lagrange multipliers. Whereas most of the current planners estimate plan quality based on the number of actions exclusively, LPG also takes into account the execution cost of an action. LPG uses this information to produce plans of good quality by minimizing an objective function which includes the overall execution cost of a plan. [11].

We modelled the coordinated-attack domain with PDDL 2.1 and used LPG to generate plans. The domain model is the following. Two users ($user1$ and $user2$) simultaneously log on to a system, both having access rights to file $f1$, which is a symbolic link to file $f2$. $User1$ executes $open\_file$ with $f1$ as argument. In the meanwhile, $user2$ removes the symbolic link from $f1$ to $f2$ and

;; Version LPG-v1.0 ;; Seed 100348977
;; Command line: ./lpg -o attack_domain.pddl -f attack_problem -n 1
;; Problem: attack_problem
;; Search time: 0.050 Parsing time: 0.010 Mutex time: 0.000
;; Actions: 5 Execution cost: 5.000 Duration: 9.000

0.001: (LOG_USER1 U2 F1 TF)[2.000] ;; cost 1.000
0.002: (LOG_USER2 U1 F1 TF)[2.000] ;; cost 1.000
2.003: (OPEN_OPENFILE U2 F1 TF)[1.000] ;; cost 1.000
3.004: (CHANGE_LINK U1 F1 TF)[1.000] ;; cost 1.000
4.005: (CLOSE_OPENFILE U2 F1 TF)[5.000] ;; cost 1.000

Time 60

Figure 7: LPG-generated attack plan for the race condition domain

creates a new symbolic link $f1$ to the target file tf, which could be */etc/shadow*, for example. As a result, $user1$ opens the target file tf which he is not authorized to access.

To succeed in the attack, attackers' actions must be synchronized. For example, action $open\_file$ must be executed first, and $change\_link$ must be executed after $open\_file$ has started, but before it finishes.

The attack domain consists of 5 durative actions which cause transitions (defined as a single action or a combination of actions) between states in the attack graph. We modelled *open_file* as a complex action consisting of two atomic actions $open\_openfile$ and $close\_openfile$. First, $open\_file$ starts with $open\_openfile$ raising a signal which is captured by *change_link* and used as a precondition for starting *change_link*. This guarantees that *change_link* can start only after *open_openfile* has started . After finishing, *change_link* raises another flag which is used as a precondition for *close_openfile* to finish. In other words, we used temporal synchronization constraints (that hold at the start and end of the action) to ensure that *change_link* is executed between the start and the end of *open_file*.

The plan generated by the LPG planner for the race-condition attack is shown in Figure 7. The PDDL 2.1 code for the attack domain, attack problem, and the output of the planner are included in Appendix A for reference.

In Figure 7, the second logging operation is executed at moment 0.002, whereas the first logging starts at 0.001. The difference is due to the fact that the actions will always be sequential unless executed in parallel processing environments.

7. CONCLUSIONS

In this paper we defined the notion of coordinated-attack graph, proposed an algorithm for efficient generation of coordinated-attack graphs, demonstrated how coordinated-attack can be used for vulnerability analysis, and discussed an implementation of a coordinated-attack graph.

Coordinated-attack graphs can facilitate an wide range of tasks, such as model checking, opponent modelling, intrusion response, sensor configuration, and so forth.

There are several open directions for future research: automatic intention recognition for coordinated attacks, adversarial planning, automatic sensor configuration for detecting coordinated attacks, etc. Solving all these problems will help improve systems security against coordinated attacks.

8. ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers for their helpful comments and suggestions.

9. REFERENCES

[1] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso. MBP: a model based planner. In *JCAI'01 Workshop on Planning under Uncertainty*, pages 71–78, 2001.

[2] C. Boutilier and R. Brafman. Partial-oder planning with concurrent interacting actions. *Journal of Artificial Intelligence Research*, 14:105–136, 2001.

[3] C. Boutilier, T. Dean, and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. In M. Ghallab and A. Milani, editors, *New Directions in AI Planning*, pages 157–172. IOS Press (Amsterdam), 1996.

[4] S. Brainov and T. Sandholm. Reasoning about others: Representing and processing infinite belief hierarchies. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 71–78, Boston, 2000.

[5] S. Braynov. On future avenues for distributed attacks. In *Proceedings of the 2nd European Conference on Information Warfare and Security (ECIW)*, Reading, UK, 2003.

[6] D. Carmel and S. Markovitch. Opponent modeling in multi–agent systems. In G. Weiß and S. Sen, editors, *Adaptation and Learning in Multi–Agent Systems*, pages 40–52. Springer-Verlag: Heidelberg, Germany, 1996.

[7] S. Eckmann, G. Vigna, and R. Kemmerer. STATL: An attack language for state-based intrusion detection, 2000.

[8] M. Fox and D. Long. *PDDL 2.1: Techical Documentation.* April 25, 2003.

[9] M. Fox and L. Long. *PDDL 2.1: An extension to PDDL for expressing temporal planning domains*. Technical Report, Department of Computer Science, University of Durham, UK, 2001.

[10] T. Garfinkel. Traps and pitfalls: Practical problems in in system call interposition based security tools. In *Proc. Network and Distributed Systems Security Symposium*, February 2003.

[11] A. Gerevini and I. Serina. LPG: a planner based on planning graphs with action costs. In *Proceedings of the Sixth Int. Conference on AI Planning and Scheduling (AIPS'02)*, pages 13–22, 2002.

[12] F. Giunchiglia and P. Traverso. Planning as model checking. In *In Proceeding of the Fifth European Conference on Planning*, pages 1–20, 1999.

[13] A. Householder, K. Houle, and C. Dougherty. Computer attack trends challenge internet security. *Security and Privacy*, 1, 2002.

[14] R. Jensen and M. Veloso. Obdd-based universal planning for synchronized agents in non-deterministic domains. *Journal of Artificial Intelligence Research*, 13:189–226, 2000.

[15] S. Jha, O. Sheyner, and J.Wing. Minimization and reliability analyses of attack graphs, 2002.

[16] S. Jha, O. Sheyner, and J.Wing. Two formal analyses of attack graphs. In *Computer Security Foundations Workshop (CSFW)*, pages 49–63, Nova Scotia, Canada, 2002.

[17] S. Jha and J. Wing. Survivability analysis of networked systems. In *International Conference on Software Engineering*, pages 307 – 317, Toronto, Canada, 2001.

[18] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and

Beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[19] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic least-commitment planning. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1073–1078, Seattle, Washington, USA, 1994. AAAI Press/MIT Press.

[20] J. Legon. *FBI seeks to trace massive Net attack*. CNN, October 28, 2002.

[21] U. Lindqvist and P. Porras. Detecting computer and network misuse through the production-based expert system toolset (p-best). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California, 1999.

[22] 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, 1998.

[23] B. Schneier. Attack trees: Modeling security threats. *Dr. Dobb's Journal*, December, 1999.

[24] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-attack graph generation tool. In *DARPA Information Survivability Conference and Exposition*, pages 146–161, Anaheim, California, 2001.

[25] S. Tzu and L. Giles. *Sun Tzu on the Art of War*. Kegan Paul Intl, 2002.

[26] G. Vigna, S. Eckmann, and R. Kemmerer. Attack languages. In *In Proceedings of the IEEE Information Survivability Workshop*, 2000.

APPENDIX

A PDDL 2.1 code for the attack domain

```
(define (domain attack_domain)
    (:requirements :strips :equality :typing :fluents :durative-actions)
(:types users files slink)
    (:predicates
        (logged1 ?u - users) (logged2 ?v - users)
        (access ?u - users ?f - files)
        (opening ?p - files)
        (opened ?q - files)
        (sym_link ?x -files ?y - files)
    )
    (:durative-action log_user1
        :parameters (?r - users ?f - files ?g - files)
        :duration (= ?duration 2)
        :condition (at start (not (logged1 ?r)))
        :effect (and
            (at end (logged1 ?r))
            (at end (access ?r ?f))
        )
    )

    (:durative-action log_user2
        :parameters (?r - users ?f - files ?g - files)
        :duration (= ?duration 2)
        :condition (at start (not (logged2 ?r)))
        :effect (and
            (at end (logged2 ?r))
            (at end (access ?r ?f))

        )
    )

    (:durative-action open_openfile
        :parameters (?r - users ?f - files ?g - files)
        :duration (= ?duration 1)
        :condition (and (at start (logged1 ?r)) (at start (access ?r
?f)))
        :effect (at end (opening ?f))
    )

    (:durative-action change_link
        :parameters (?l - users ?s -files ?t - files)
        :duration (= ?duration 1)
        :condition (and (at start (logged2 ?l)) (at start (opening ?s)))
        :effect (at end (sym_link ?s ?t))
    )

    (:durative-action close_openfile
        :parameters (?r - users ?f - files ?g - files)
        :duration (= ?duration 5)
        :condition (and (at start (logged1 ?r)) (at start (access ?r ?f))
            (at start (opening ?f)) (at start (sym_link ?f ?g)))
        :effect (and
            (at end (opened ?g))
        )
    )
)
```

PDDL 2.1 code for the attack domain

```
(define (problem attack_problem) (:domain attack_domain)
(:objects
        u1 - users
        u2 - users
        f1 - files
        tf - files
)
(:init
        (not (logged1 u1))
        (not (logged1 u2))
        (not (logged2 u1))
        (not (logged2 u2))
)
(:goal
        (opened tf)
)
(:metric minimize (total-time)) )
```