Dependency Relation Based Vulnerability Analysis of 3G Networks: Can It Identify Unforeseen Cascading Attacks ?

Kameswari Kotapati, Peng Liu, and Thomas F. LaPorta
kotapati@cse.psu.edu, pliu@ist.psu.edu, and tlp@cse.psu.edu
The Pennsylvania State University, University Park, PA 16802

Abstract

Cascading attacks pose a new threat to the third generation (3G) wireless telecommunications network. These attacks are dangerous and difficult to detect due to their remote *far-reaching* effects. To automate the accurate detection of these attacks and their remote effects, we developed a *telecommunication specification* based toolkit called the *Advanced Cellular Network Vulnerability Assessment Toolkit - aCAT*. aCAT is unique due to the incorporation of 3G network specific *dependency model, infection propagation rules*, as well as *expert knowledge*. These features allow aCAT to accurately and exhaustively identify cascading attacks and their remote effects. aCAT illustrates the types of cascading attacks that may be derived from the specifications, and showcases its utility in uncovering these attacks.

1 Introduction

Anytime anywhere accessibility, seamless roaming, inexpensive handsets with sophisticated applications, and Internet connectivity are only a few of the myriad reasons that have made the 3G network an indispensible part of daily lives of millions of people. The huge success and the popularity of these networks may not only be attributed to their support for mundane communication but also to the many life and mission critical services such as E-911 supported by the 3G network. Accordingly, from the security point of view, 3G networks are extremely attractive targets to the adversary.

3G networks are attractive to adversaries not only due to their large subscriber base (which the adversary could target in an attack) but also because of their highly vulnerable nature, making it easy for the adversary to launch an attack. There are two main reasons that account for the highly vulnerable nature of the 3G network. First being that, 3G networks were built for performance and service, security was added-on, as and when required. To this day there are many vulnerabilities that could be effortlessly exploited to cause havoc on infrastructure and services. Secondly, introduction of Internet connectivity to 3G networks imports not only the high speed capabilities of the Internet but also its inherent vulnerabilities. Thereby making possible cross infrastructure cyber attacks - a new breed of attacks, where attacks against the 3G network may be lauched from the Internet.

Although the number of vulnerabilities in the 3G network are innumerable, existing vulnerability assessment solutions for the 3G network are very limited in number. Although many 3G attack scenarios have been identified (e.g., in [1, 2, 5, 11, 14, 16, 17, 22, 23, 25, 26]) and a few 3G threat taxonomies have been proposed (e.g., in [21,26,41]), prior research has not developed any systematic yet automatic techniques for 3G network vulnerability assessment.

As a result, 3G network designers, developers, vendors, and operators are still unaware of many vulnerabilities. Many unforeseen attacks enabled by exploiting these vulnerabilities can be unexpected, undetected and hence very destructive. Also these attacks may be subtle in nature with many far-reaching effects. Such subtle attacks may neither be identified by existing vulnerability assessment tools, nor be identified by 3G engineers or operators until they cause some serious effects.

As an illustration of such a subtle attack is the *alerting attack* (presented in Section 2.2). This attack allows an adversary (e.g., an terrorist) to subtly prevent a police officer (or an emergency healthcare provider) from receiving an E-911 call in a undetected manner, using subtle attack actions that have many remote far-reaching consequences. Accordingly, we call attacks like the alerting attack as *cascading attacks*. Cascading attacks are so named because local effects of corrupt data items propagates or cascades to data items on remote service nodes through vehicles such as signaling messages, cached data items, and shared databases.

To detect such subtle cascading attacks and expose vulnerabilities, we developed **aCAT** (Advanced Cellular network vulnerability Assessment Toolkit), a novel 3G network vulnerability assessment toolkit. To our best knowledge, aCAT is the first toolkit that can automatically identify subtle cascading attacks in a systematic way. (Note that CAT [20] is a very preliminary version of aCAT.)

aCAT incorporates a unique network dependency model, infection propagation rules, and a small amount of expert knowledge to expose cascading attacks. aCAT detects these attacks based on user input, which are 3G data item(s) that are either directly corrupted by the adversary (`seeds`) or the ultimate data item(s) corrupted as a remote cascading effect of adversary action (`goals`), and system input that comprises of telecommunication specifications.

Telecommunication specifications are defined by the Third Generation Partnership Project (3GPP) and are available at no charge at [6]. These specifications detail the functional behavior and not the implementation structure of the 3G networks. Also they are written using simple flow-like diagrams called the Specification and Description Language (SDL) [15]. Henceforth we will refer to these specifications as SDL specifications. Equipment and service providers use these SDL specifications as the basis of their service implementations. These specifications may also be used by adversaries (possibly disgruntled employees) to develop debilitating attacks.

When an (unforeseen) attack is identified by aCAT, aCAT portrays the attack, including its origins and remote cascading effects, in a user friendly attack graph format. Attack graphs illustrate ways in which an attack may be launched, thereby exposing all vulnerabilities. This allows network designers to determine vulnerable points in the network that need protection.

We have used aCAT in assessing the vulnerability of several key services offered by the 3G networks, and found that aCAT *can* identify interesting and unforeseen cascading attacks which are subtle and hard to be identified by other means. These newly identified cascading attacks include the Alerting Attack, Power-off Power-on Attack, Mixed Identity Attack, Call Redirection Attack,and Missed Calls Attack.

aCAT is the first step in conducting automatic and systematic 3G network vulnerability assessment. The merits of aCAT are summarized as follows.

- aCAT is the first tool that can identify *unforeseen* cascading attacks.
- aCAT is *automatic*: given the required input, the vulnerability assessment process does not require any human intervention.
- aCAT uses *standard* specification provided by the 3GPP as the primary input. As a result, aCAT is *agnostic* to specific 3G network implementations/configurations, its method is *practical*, and it has universal *applicability*.

- Using comprehensive specifications and a comprehensive 3G network dependency model, aCAT can perform *systematic* vulnerability assessment.
- aCAT performs *fine-grained* vulnerability assessment at the message level, the process level, and the data item level.

The rest of the paper is organized as follows. In Section 2, we give an overview of the problem and our solution. In Section 3, we detail our attack graph and its features and in Section 4, we present the the aCAT models. In Section 5, we detail our algorithms and in Section 6, we present some interesting attacks detected by aCAT. In Section 7, we describe the performance of aCAT and in Section 8, we present some observations. In Section 9, we discuss the related work and conclude the paper in Section 10.

2 Overview

In this section, we provide an overview of 3G networks, the cascading attack and the method used to identify these attacks.

2.1 Call Delivery Service in 3G Networks

In this section, we use the *call delivery service* to illustrate some core components (or building blocks) of 3G networks. The call delivery service is a basic 3G service. It is used to deliver incoming calls to any subscriber with a 3G enabled mobile device regardless of their location. Although aCAT is a generic tool applicable to all 3G services, throughout the paper we will continue to refer to this service as the primary attack target to self-contain our presentation.

3G networks provide service by the exchange of signaling messages among its various servers called service nodes. As shown in Figure 2, the service nodes in the circuit switched domain of the 3G network are the Home Location Register (HLR), the Visitor Location Register (VLR), the Mobile Switching Center (MSC) and the Gateway Mobile Switching Center (GMSC). 3G services are provided across interconnected networks that cover certain geographical areas. Accordingly, every subscriber is assigned a *home network* from where they may roam to other *visiting networks*.

The home network stores the profile and current location (pointer to VLR) of all subscribers assigned to it in the HLR. In addition, each network administrative area is assigned a VLR. The VLR stores temporary data of subscribers currently roaming in its assigned area; this subscriber data is received from the HLR of the subscriber. Every VLR is typically assigned a MSC that acts as an interface between the radio system and the fixed network, and handles circuit switched services for subscribers currently roaming in its area.

When a subscriber makes a call, the call (signaling message `IAM`) is sent to the nearest GMSC which is in charge of routing calls and passing voice traffic between disparate networks (refer Figure 1). Each signaling message contains data items used to invoke functions at the destination service nodes. For example, the `IAM` signaling message contains the 'called number' data item and is used to invoke the function that finds the assigned HLR (home network) of the called party, at the GMSC. The GMSC signals the HLR of the incoming call using the signaling message `SRI`. The `SRI` message contains data items such as the *called number* and the *alerting pattern*. The *alerting pattern* denotes the pattern (PACKET SWITCHED DATA, SHORT MESSAGE SERVICE OR CIRCUIT SWITCHED CALL) that is used to alert the called mobile subscriber.

As the HLR is aware of the location of the called subscriber, it requests the corresponding VLR for a roaming number (message PRN) by which to route the call. It then downloads the incoming call profile of the subscriber (including *alerting pattern*) to the VLR. The VLR assigns a roaming number for routing the call and passes it on to the HLR (message PRN_ACK) which forwards it to the GMSC (message SRI_ACK). The GMSC uses this roaming number to route the incoming call (message IAM) to the MSC where the subscriber is currently visiting. This MSC requests the incoming call profile for the called subscriber (message SIFIC) from the VLR and receives the profile (including *alerting pattern*) in the Page MS message. The MSC uses the *alerting pattern* in the profile to determine the *page type* which is the manner in which to alert (message Page) the mobile station. Thus subscribers receive incoming calls irrespective of their locations in the network.

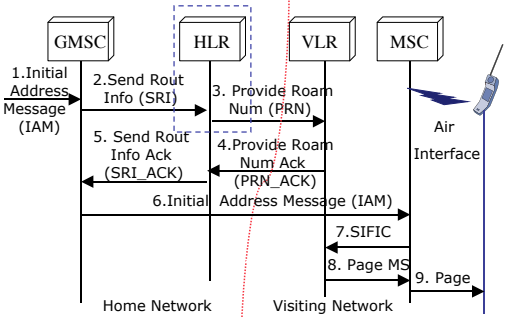

Figure 1: Call Delivery Service

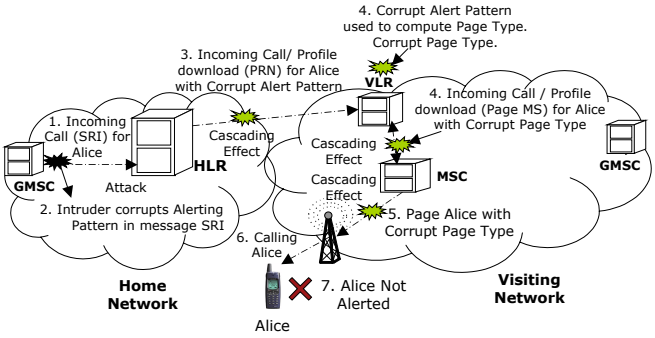

Figure 2: Alerting Attack

2.2 Example: Cascading Attack

In this section, we present a cascading attack called the *Alerting Attack*, on the call delivery service discussed in the previous section. This attack was discovered by aCAT using only the SDL specifications. This attack highlights the fact that an adversary with access to the SDL specifications can devise cascading attacks without much effort.

In this attack, the adversary targets the data item *alerting pattern* in the SRI signaling message generated by the GMSC. According to the specifications, the *alerting pattern* may take values PACKET SWITCHED DATA, SHORT MESSAGE SERVICE, OR CIRCUIT SWITCHED CALL. Here, the adversary switches the value of *alerting pattern* from the required value of CIRCUIT SWITCHED CALL to the SHORT MESSAGE SERVICE. Since the *alerting pattern* is corrupted at the time of assignment and the corrupt value is a system acceptable value albeit an incorrect value, corruption not only remains undetected but also propagates to remote areas of the network.

The corrupt *alerting pattern* in the SRI message is passed on to the HLR. The PRN signaling message sent by the HLR to the VLR contains the corrupt *alerting pattern*. The VLR uses the corrupt *alerting pattern* to compute *page type*. *Page type* is assigned a value corresponding to the value of *alerting pattern*. If the *alerting pattern* is set to SHORT MESSAGE SERVICE, the *page type* to be assigned the value SMS. is automatically assigned a value corresponding to the incorrect value and is therefore corrupt. This happens automatically due to normal network operation and without any adversary intervention.

The VLR outputs message Page MS to the MSC with corrupt data item *page type*. The MSC outputs the signaling message Page to the base station (BSS) with corrupt *page type* which is then

broadcast in the `Page MS` message. If the *page type* is incompatible with the type of call (PACKET, CIRCUIT SWITCHED, etc.) expected by the mobile station, the call session may not be received. This illustrates how corrupting a data item at one point in the network such as the GMSC, propagates corrupted data across the network, such as to the BSS during normal network operation.

This attack shows that data corruption in a 3G network has many indirect and remote far reaching *cascading effects* due to normal network operation. Cascading effects give rise to the *chaining* phenomena. A *chain* is a sequence of k corrupt data items (also called chain items). The first chain item is the `seed` (data directly corrupt by the adversary), the second chain item is derived from the first chain item and so on. The final chain item is called the `goal` because it directly serves the adversary's intent. In the case of the alerting attack, the `seed` is the *alerting pattern* and the `goal` is the *page type*.

The `goal` item is usually defined by the effect of the attack on (a) the network operation and; (b) the target subscribers. Hence, a goal item has the following two properties: (1) it is the direct cause the for network mis-operation and; (2) the caused mis-operation directly effects the target subscriber.

In real life, adversaries (e.g. disgruntled employees) may gain access to improperly guarded central offices [39], which house telephony equipment. Details of these offices including their addresses, photographs and manuals to telephony switches may be obtained at websites such as [38, 39]. Many telephony switches are based on UNIX-like operating systems with software comprised of several million lines of C code. These switches may be configured to function as a GMSC. Using switch manuals found on websites such as [38], one can learn how to log-on to the switch, issue commands to get permissions to modify data items and make changes to software. The adversary can corrupt the *alerting pattern* as described earlier, using commands similar to the following. Based on the configuration of the switch the syntax of the commands may vary.

if(`alerting_pattern` **.is.** CIRCUIT SWITCHED CALL) **then**
    `alerting_pattern` = SHORT MESSAGE SERVICE;
end – if

2.3 Manual attack detection

Using SDL specifications network engineers can manually predict attacks. This is because specifications are easy to read, they contain (1) signal flow graphs such as shown in Figure 1, that present a high level picture of network functionality; and (2) flow chart like SDL diagrams as shown in Figure 6b and 6c, that present minute details of the network functionality. Minute details of network functionality include input message received, data items contained in them, functionality invoked by these data items, and the output messages generated. An example of minute network functionality is Figure 6c, which shows the HLR service node, responding to the incoming message `SRI` (denoted by dotted line in Figure 1). Specifications provide diagrams similar to Figure 6c for each and every service node functionality.

Using these diagrams, network engineers can manually trace data corruption across service nodes and messages such as in the alerting attack case. However, a 3G network is comprised of hundreds of services such as the call delivery, each service involving hundreds of messages and services nodes, which in-turn are comprised of thousands of state machines. In such cases manually tracing data corruption through service nodes and messages may be tedious and error prone. Hence, we need an

efficient and speedy method devoid of human error with a fixed set of rules to track corruption across thousands of service nodes. aCAT is developed for these reasons. It tracks data corruption propagation efficiently by incorporating SDL specifications, network dependency model, and a set of infection propagation rules. In the next section, we explain the various features of aCAT.

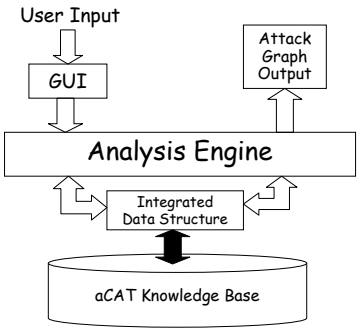

Figure 3: Architecture of aCAT

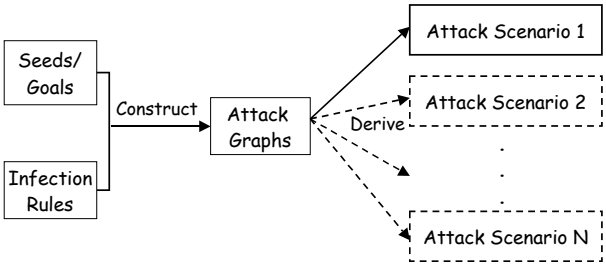

Figure 4: Work Flow of Vulnerability Assessment

2.4 Automated Solution- aCAT

aCAT is implemented using the Java programming language. It is made up of a number of subsystems (as shown in Figure 3). The *aCAT-knowldege base* contains the 3G network knowledge. The major portion of this knowledge is obtained from SDL specifications. A minor portion of this knowledge is received from an expert due to limitations in SDL. Data in the knowledge base is formatted using the unique 3G network dependency model detailed in Section 4. The *integrated data structure* is similar to that of the knowledge base; it holds intermediate vulnerability analysis results.

a number of areas hence we take the help of an expert to compensate for these limitations.

The *GUI subsystem* takes in user input in the form of `seeds` or `goals`. The *analysis engine* comprises of algorithms (forward, reverse, and combinatory). These algorithms are incorporated with the 3G specific infection propagation rules that identify corruption propagation. Using these infection propagation rules and user input the analysis engine captures the attack in the form of an *attack graph*. We use the term 'attack graph', as our graph is similar to the Internet based attack graphs [7, 34]. However, there are some major differences that are explained in Section 9.

Our attack graphs are user friendly and show the propagation of corruption through the network. The attack graph explains specifically it can be said that the 3G attack graphs show the network effects of the attack. Using these attack graphs, realistic attack scenarios may be derived. Attack scenarios explain the effect of the attack on the subscriber in a realistic setting. Each attack graph may have multiple interpretations and give rise to multiple scenarios. Each scenario gives a different perspective on how the attack may effect the subscriber. This vulnerability assessment work flow is illustrated in Figure 4. In the next section, we detail features of the attack graphs and attack scenario derivation.

3 3G Attack Graph and Interpretation

In this section, we present the 3G attack graph and its interpretation.

3.1 3G Attack Graphs

Cascading attacks may also be defined as network state transitions caused by an adversary's action, where the final transition results in the adversary achieving a goal. aCAT deduces possible attacks on 3G networks and represents them in the attack graph format. A 3G network specific attack graph may be defined as a network state transition showing the paths through a system starting with the conditions of the attack, followed by attack action and ending with its cascading effects. The 3G network state may be defined as the collective state of all its components.

Figure 5 shows the attack graph output produced by aCAT. This attack graph output is in telecommunication terminology and corresponds to the previously described alerting attack. For description purposes, the attack graph has been divided into levels and assigned node labels.

Nodes represent states in the network with respect to the attack and may be broadly classified as *conditions, actions, and goal(s)*. Nodes at the lowest level typically correspond to the conditions that must exist for the attack to occur, such as *adversary's physical access, target and existing vulnerability*.

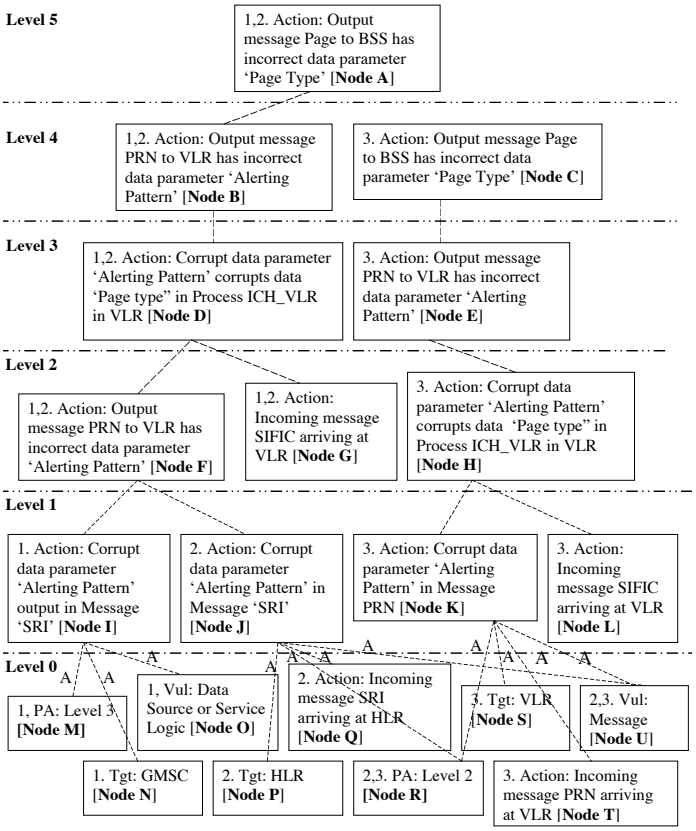

Figure 5: Attack Graph for Attack Alerting

The adversary may have any of the following three levels of *physical access*: (1) access to the air interface with the help of a physical device; (2) access to links connecting central offices; and (3) access to the service nodes. In the alerting attack, the adversary has access to the GMSC, i.e. level 3 physical access which is represented in the attack graph by Node M. The adversary's target is always a service node such as the GMSC and is represented by Node N. The adversary may take advantage of

vulnerabilities such as the service logic (detailed shortly in the threat model in Section 4.3) represented by Node O. Our attack graphs show all the possible conditions for an attack to happen, i.e., we not only see the alerting attack due to corruption of service logic at the GMSC, but also other possibilities such as the corruption of signaling message `PRN`.

Nodes at higher levels are actions that typically correspond to effects of the attack propagating through the network. Effects typically include propagation of corruption between service nodes, such as from HLR to VLR (Node F), propagation of corruption within service nodes, such as *alerting pattern* corrupting *page type* (Node D), and so on. Actions may further be classified as *adversary actions, normal network operations or normal subscriber activities*. Adversary actions (Node I) include the insertion, corruption or deletion of data, signaling messages and service logic. Normal network operations (Node L) include sending and receiving signaling messages. Subscriber activity may include updating personal data or initiating service.

Goal nodes typically occur at the highest level of the attack graph. They indicate corruption of the `goal` items due to the direct corruption of `seeds` by the adversary (Node A).

In our graph, edges represent network transitions due to both normal network actions and adversary actions. Edges help show the global network view of adversary action.This is the uniqueness of our attack graph. Transitions due to adversary action are indicated by an edge marked by the letter 'A' (edges connecting level 0 and level 1).

3G attack graphs are obtained by pruning and merging *attack trees* constructed by aCAT. Hence individual attack trees are part of the attack graph shown in Figure 5. Each attack tree shows the attack effects due to corruption of a seed at a specific network location. In the graph, trees are distinguished by the tree numbers assigned to its nodes. For example, all the nodes marked with number 2 belong to Tree 2 of the graph.

Tree 1 shows the propagation of corruption due to the corruption of the seed *alerting pattern* in the GMSC. Tree 3 shows the propagation of the alerting attack due to the corruption of the seed *alerting pattern* in the signaling message `SRI` ending in output message `Page MS` containing the incorrect *page type* `goal` item. Nodes P, Q, R and U represent the conditions for the attack and Node J shows adversary action. Nodes F, D, and B show the propagation of the attack and Node A depicts the goal node of the alerting attack. These trees show that the vulnerability of the 3G network is not limited to one place, but can be realized due to the corruption of data in many network locations.

Some nodes in the graph belong to multiple trees. Tree numbers are also used to distinguish between *AND* and *OR* nodes at a level in the graph. Nodes at a particular level with the same tree number(s) are *AND* nodes. For example, at Level 2, Node F and Node G are *AND* nodes; both must occur for Node D at Level 3 to occur. Nodes at a particular level, with different tree numbers and edges connecting to the same node at a higher level are *OR* node. For example, at Level 1, Node I and Node J are *OR* nodes; either one of the nodes may occur for Node F at Level 2 to occur. Other nodes have no relation. For example, at Level 2, Node H and Node G have no relation.

This attack graph format is well-suited for telecommunication networks because data corruption propagates through the network in various forms during the normal operation of a network; thus, an attack that corrupts a data item may manifest itself as the corruption of a different data item in a different part of the network only after some network operations take place.

3.2 Attack Scenario Derivation

This section explains the principles involved in the derivation of realistic attack scenarios from attack graphs in telecommunication semantics.

Step 1: End User Effect: Goal node(s) are used to infer the end effect of the attack on the subscriber. In the alerting attack, the goal nodes are Node A at Level 5, and Node C at Level 4. According to the goal node, the `Page` message to the BSS has incorrect `goal` item 'page type'. The `Page` message is used to inform subscribers of the arrival of incoming calls and 'page type' indicates the type of call. 'Page type' must be compatible with the subscriber's mobile station or else the subscriber is not alerted. From the goal node it may be inferred that Alice, a subscriber of the system is not alerted on the arrival of an incoming call and hence does not receive incoming calls.

Step 2: Origin of Attack: Nodes at level 0 indicate the origin of the attack, and hence the location of the attack may be inferred. The alerting attack may originate at the following locations: signaling messages `SRI`, `PRN`, the service node VLR, or the HLR.

Step 3: Attack Propagation and Side effects: Nodes at all other levels show the propagation of corruption across the various service nodes in the network. In the alerting attack, from the other levels it may be inferred that the `seed` is the alerting pattern and the attack spreads from the HLR to the VLR and from the VLR to the MSC. We found that some non-goal nodes (that indicate the propagation of corruption) also indicate side effects on the user in addition to the goal node that shows the final effect on the user. Examples of these side user effects are provided in the Section 6.

Attack Scenario: Using the above guidelines the following attack scenario may be derived. Trudy, the adversary corrupts the *alerting pattern* of Alice, the victim, at the `SRI` message arriving at the HLR.

When there is an incoming call for Alice, her profile is downloaded from the HLR to the VLR and from the VLR to the MSC. The call proceeds as usual but Alice is not paged as required. Hence Alice's mobile station cannot detect the incoming call and the call is missed. This attack is subtle to detect because 3G administrators find that the network processes the incoming call correctly and that the subscriber is alerted correctly. They may not find that this alerting pattern is incompatible with the mobile station itself. This attack is illustrated in Figure 2.

Using aCAT we have discovered a set of interesting attacks besides the alerting attack which we present in Section 6. In the next section, we present the network dependency model and infection propagation rules used by aCAT.

4 The aCAT Model

In this section, we present the various model and rules used in the development of aCAT.

4.1 3G Network Model

Since aCAT uses SDL specifications to derive attacks, we model 3G networks using SDL notations. In our model, the 3G network is a set of concurrently running service nodes called *blocks* that communicate by exchanging signaling messages. A block has two types of components: data items and concurrently running processes. A block B_i is associated with the following 4 types of data items: (1) data *parameters* contained in signaling messages to and from the block; (2) updatable data items stored in an associated database (called a data source); (3) cached read-only data items; and (4) other temporary local variables used in processing.

In a block, *processes* use *service logic* to perform functions. The service logic in every process includes (1) *process functionality* used to compute data items and change the data associated with the block; (2) *database transactions* between processes and the associated database; and (3) *invocation* of other processes. Processes may be broadly classified as Mobility Management, Call Handling, Operations and Maintenance, Fault Recovery, Handover and Subscriber Management. A process may also be defined as a communicating extended finite state machine (CEFSM) which is a special case of the extended finite state machine (EFSM). Figure 6a shows the CEFSM of a process in the HLR (indicated by dotted rectangle in Figure 1).

4.2 SDL Model

SDL is a graphical object-oriented, formal language designed for the specification of event-driven, real-time, concurrent distributed systems interacting with discrete signals. In basic SDL, the system description is hierarchically structured to describe the local and remote behavior of telecommunication systems, as Systems, Blocks and Processes. The SDL System can be mapped to the 3G network; the SDL Block and Process may be mapped to the block and process defined in the 3G network model of Section 4.1, respectively.

Processes are the basic functional units of SDL systems. Specifications represent all network functions using the SDL process. Figure 6b shows the graphical syntax of an SDL process. Figure 6c shows the actual SDL fragment for the 3G process state transition diagram of Figure 6a. On comparison of Figure 6a and 6c it is obvious that the SDL process is similar in structure to the 3G process CEFSM. Hence the mapping from the 3G CEFSM process to SDL process is one-to-one. The SDL diagrams in Figures 6b & 6c are representative of all the diagrams used in SDL specifications.

4.3 Threat Model

Our work is focussed on the remote cascading effects that occur due to propagation of corrupt data items across (multiple) service nodes or blocks. Hence our threat model includes any attack actions (such as various buffer overflows) that may produce a seed. Readers interested in how these actions may be taken may refer to [21].

Based on their effects, we classify the relevant attack actions as those that corrupt (1) signaling messages; (2) caches; (3) database records; (4) local variables; and (5) service logic; distorted service logic will indirectly corrupt a message, a database record, or a local variable. Refer to Figure 7 for the relationship between items that may be effected by the attack. In real life, these attack actions include (1) social engineering schemes that give adversaries access to service nodes and allow them to corrupt service logic; and (2) exploitation of software vulnerabilities that may overload the switch resulting in buffer overflow. Besides cascading attacks, there are a variety of other 3G attacks, as summarized in [21]; however, they are out of the scope of this work.

4.4 Network Dependency Model and Infection Propagation Rules

In principle, cascading attacks are the result of propagation of corruption between 3G components due to dependencies (relationships) that exist between these components. Hence to uncover these attacks, we define a network dependency model to clearly identify relationships between these 3G components. We also use this network dependency model to define a set of infection propagation (IP) rules (detailed in Table 1) to capture the propagation of corruption.

Our network dependency model (illustrated in Figure 7) specifies the basic 3G components as the data items, signaling messages, and service logic. In our first IP rule we define what constitutes corruption of these basic components. Subsequently, we categorize dependencies as being within a block (*intra-block*) or between blocks (*inter-block*). In particular, our second IP rule defines that corruption due to inter-block dependencies can exist only due to the exchange of signaling messages. Figure 7 illustrates this inter-block dependency in signaling messages M_1, M_2, and M_3. Message M_1 contains corrupt data item d_A indicated by d_A^* and hence spreads corruption from block B_i to B_j.

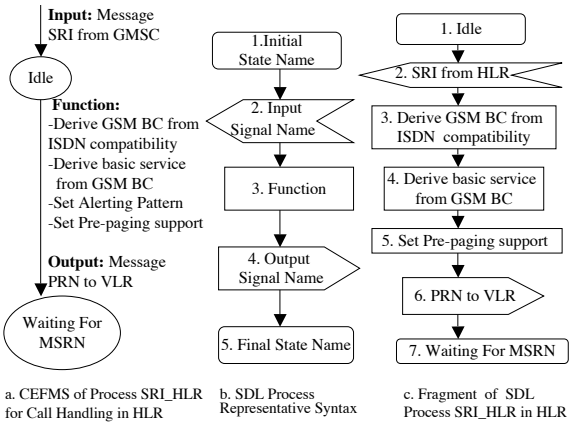

Figure 6: Example of CEFSM and SDL

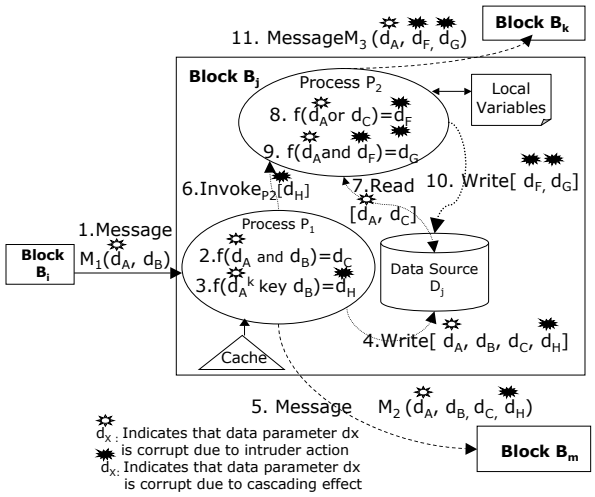

Figure 7: Network Dependency Model

We further classify *intra-block dependency* as either (1) *derivative dependency* that defines relationship between data items in a block; or (2) *process to data source dependency* that defines relationship between a process and the data source within a block; or (3) *inter-process dependency* that defines relationships between processes within a block. Intra-block dependency is typically invoked in response to an inter-block dependency i.e. the arrival of a signaling message and the use of data items in these messages as input to the intra-block dependency (e.g. P_1). In summary, an intra-block dependency is invoked by an inter-block dependency.

Derivative dependencies cause the spread of corruption between data items in a block due to the *process functionality used in computing these data items*. The process functionality uses 'n' data items (received in messages or owned by the block) as input ($\texttt{fn}^{\texttt{Input}}$) with derivative dependency operators (`Dep_Operator`) to derive output data items ($\texttt{fn}^{\texttt{Output}}$). The derivative dependency operators are AND, OR or KEY. Our IP rule 3 details the spread of corruption due to each derivative dependency operator. In the following, we also use examples to explain the propagation of corruption due to each derivative operator .

Corruption propagation is exhibited by *AND dependency operator* in process P_1 of Figure 7 where, d_A and d_F are both corrupt input items to an AND derivative operator used to compute and hence corrupt d_G. Process P_2 exhibits corruption propagation due to the *OR dependency operator* where data items d_A and d_C are used as input to compute output item d_F. d_F is corrupt as a result of corruption of d_A alone. Figure 7 shows P_1 using data items d_A and d_B as the input key to a *KEY operator* to retrieve d_H. d_H is incorrect as a result of corrupt d_A being used as the retrieval key.

Process to data source dependency exists between processes and the data sources owned by a

Table 1: Infection Propagation Rules

No.	Corruption	Infection Propagation Rule
1.	Basic Components	A data item d_a of correct value u at any instant of time t is said to be corrupt iff there exists a value v, of d_a at the same instant of time t where $u \neq v$. A signaling message is said to be corrupt iff there exists a data item contained in the message where the data item has been corrupted directly by an adversary or indirectly due to cascading effects. The service logic of a process is corrupt iff there exists a set of input data items with correct original value and the output item computed with the service logic is corrupt.
2.	Inter-Block Dependency	Corruption spreads between blocks iff there exists a corrupt signaling message propagating between the respective blocks.
3.	Derivative	The output data item in an AND derivative dependency is corrupt iff all 'n' input data items are corrupt. The output data item in a OR derivative dependency is corrupt iff any one of the 'n' input data items are corrupt. The output data item in a KEY derivative dependency is incorrect iff the key (input item) used to retrieve the output item is corrupt.
4.	Messages	The output message generated by the process is corrupt iff either one of the following is true: 1. The service logic of the process is corrupt. 2. The service logic of the process is not corrupt but either one of the following is true: A. Input signaling message is corrupt. B. Data source owned by the block is corrupt.
5.	Chaining	Chains in cascading effects occur iff one of the following are true: 1. Input data items are corrupt and the AND, OR or KEY dependency results in corrupt output data item(s). 2. Service logic of the process is corrupt resulting in corrupt output data items. 3. Service logic of the process is corrupt resulting in incorrectly written data sources. 4. Service logic of the process is corrupt and invokes incorrect processes. 5. Service logic of the process is corrupt and produces corrupt signaling messages.

block. It is due to the process reading [`Read`()] and writing [`Write`()] data items to data sources. Data items used in transactions with the data sources may be received in a message or owned by the block itself. Hence a *Process to data source* dependency can cause corruption to propagate from a process to a data source through read's and write's. This is illustrated in Figure 7 by process P_2 reading corrupt item d_A and d_C from the associated data source and writing corrupt items d_F and d_G. *Inter-Process* dependency occurs between processes in a block due to process invocation [`Invoke`()] of other processes. Corruption propagates in a inter-process dependency when processes invoke other processes using corrupt data as input in the invocation. This is illustrated in Figure 7 by process P_1

invokes process P_2 using incorrect data d_H within block $\mathtt{B_j}$.

As messages cause corruption to propagate to remote service nodes, our IP rule 4 takes the various classes of intra-block dependency into consideration in defining the causes for corruption of an output message. Finally, all cascading effects give rise to chains. By using our IP rule 5, these chains can be accurately derived.

4.5 Propagation Model

We also classify the type of chains. A *chain* is a result of the cascading effect, and it is comprised of a sequence of corrupt chain items, where each chain item is derived from the previous one. Although chains can be of many types, we broadly classify them as *linear* or *branching*. In the *linear chain*, a single corrupt chain item is sufficient to corrupt the next chain item. For example, a single corrupt chain item d_1 leads to corruption of chain item d_i and so on ($d_1 \rightarrow d_i \rightarrow \ldots \rightarrow d_n$). The *branching chain* is specifically caused by the AND derivative dependency, here all input items in the AND dependency must be corrupt to corrupt the next chain item. For example, both the chain items $d_1,\ d_2$ in the AND input lead to corruption of d_3, the next chain item (d_1 & $d_2 \rightarrow d_3$; d_3 & d_4 & $d_5 \rightarrow \ldots \rightarrow d_n$).

5 Algorithms

In this section, we present our algorithms and the aCAT knowledge base used by these algorithms to build attack graphs.

5.1 aCAT Knowledge Base

The *aCAT knowledge base* contains knowledge of the 3G service(s) such as the call delivery service, whose vulnerability is subject to analysis. The database is populated using SDL specifications for call delivery service [3] and expert knowledge. We have also added a minor amount of expert input, in those areas where SDL specification is ambiguous and lacking. Knowledge from the specifications and expert is formatted with the network dependency model for application of IP rules.

Table 2: aCAT Knowledge Base: Table-1: Signaling Messages

Message Name	Data Items
SRI	Mobile Station International ISDN Number (MSISDN), Alerting Pattern, CUG interlock, CUG outgoing message, ISDN BC, ISDN LLC, ISDN HLC, GMSC Pre-paging support
PRN	International Mobile Subscriber Identity (IMSI), Mobile Station International ISDN Number (MSISDN), GSM bearer capability, ISDN BC, ISDN LLC, ISDN HLC, Alerting Pattern, Pre-paging supported

The tables in the aCAT knowledge base are shown in Table 2, and Table 3. Table 2 shows the structure aCAT knowledge base Table-1 and some sample data stored in the table. This table stores the signaling message information information and data parameters contained by them. Sample data shows the data parameters contained in `SRI` and `PRN` signaling messages. This table aims to captures all the data items that may be used to propagate corruption between blocks i.e inter-block dependencies.

Table 3 shows the structure of the CAT knowledge base Table-2, and some sample data stored in this table. This table stores the information of the basic functional unit of the SDL system, i.e the process. Every column in this table corresponds to every component in a SDL Process shown in Figure 6b. Thus, it captures all the aspects of a SDL Process, i.e. the initial, and final states, input and output messages, and functions. This table aims to captures all aspects of process functionality that may be used to propagate corruption within a block, i.e. intra-block dependencies.

On close observation, it can be seen that the sample data contained in this table is a direct translation of the SDL fragment shown in Figure 6c with the network dependency format applied to it. For example, the function *Derive GSM BC from ISDN compatibility information* is replaced by the OR derivative dependency `f(ISDNBC | ISDN LLC | ISDN HLC)= GSM Bearer Capability` (first part); the function *Derive basic service from GSM BC* is replaced by an KEY derivative dependency `(GSM Bearer Capability~)=Basic Service` (second part); and the function *Set Pre-paging support* is replaced by an AND derivative dependency `(HLR Pre-paging support` $\wedge$ `GMSC Pre-paging support` $\wedge$`)=Pre-paging support` (fourth part). These functions are a direct translation from the SDL.

Table 3: aCAT Knowledge Base: Table-2: Processes

Process Name	Initial State	Input Message	From Block	Function	Output Message	To Block	Final State
SRI_HLR	Idle	SRI	GMSC	(1) (ISDN BC ‖ ISDN LLC ‖ ISDN HLC)=GSM Bearer Capability (2) (GSM Bearer Capability ~)=Basic Service (3) (MSISDN ~)=IMSI; (MSISDN ~)=HLR Pre-paging support; (MSISDN ~)=MSC#,VLR#, LMSI (4) (HLR Pre-paging support ∧ GMSC Pre-paging support ∧) =Pre-paging support	PRN ← **Expert Input**	VLR	Wait for MSRN

We also added some expert input to the table (third part). For example, the MSISN is used to retrieve a number of data parameters, hence we added the following KEY derivative dependencies `[(MSISDN ~)=IMSI]`, `[(MSISDN ~)=HLR Pre-paging support]`, `[(MSISDN ~)=MSC#, VLR#, LMSI]`.

In addition to the derivative dependencies presented in Table 2 of the aCAT knowledge base, the function column of Table 2 may also contain the following: (a) *process and data source dependencies* represented by reads `Read(`d_x`)` and writes to the database `Write(`d_x`)`; and (b) *inter-process dependencies* represented by invocation to other processes `Invoke(`d_x`)`.

Thus using Table 1 in the aCAT knowledge base, we can identify the inter-block dependencies and by using Table 2 we can identify all the intra-block dependencies in the network dependency model. As illustrative example is the alerting attack, the `SRI` signaling message arrives at the HLR with a corrupt *alerting pattern*. The aCAT knowledge base captures this information in Table-1. On observing the

function column of Table-2 in aCAT knowledge base, we can see that the *alerting pattern* does not corrupt any other data parameters, i.e. does not cause intra-block corruption. But Table-1 shows that the output message PRN contains the same *alerting pattern* and hence captures inter-block corruption. Inter-block corruption occurs as a corrupt *alerting pattern* contained in the message `PRN` propagates from the HLR to the VLR. Thus by observing data parameters, contained in messages and processes flow of corruption may be traced.

5.2 Algorithms

We have designed forward, reverse and combinatory algorithms to detect attacks and build attack graphs in an exhaustive depth first search. These algorithms (forward, reverse and combinatory) are incorporated with the IP rules, and are implemented using the Java programming language.

As shown in Figure 8, the forward algorithm takes as input a single *seed*, detects the rest of the chain items till the goal, and builds the attack graph bottom-up. The reverse algorithm takes as input a single *goal*, detects the previous chain items till the seed, and builds the attack graph top-down. The combinatory algorithms takes as input multiple *seeds* or *goals* and searches for chain items that denote the combined effect of the input.

All of the algorithms are *exhaustive* as they check every process in the block (indicated by line 3 in Figure 8), and *succinct* as only those states that reach the goal/seed are added to the attack graph. We only discuss the forward algorithm (shown in Figure 8) in detail due to space limitations.

Our forward algorithm works by (1) locating the origin of the attack; (2) finding the next chain item which is comprised of building the attack graph bottom-up; and (3) pruning the graph. Our algorithm assumes that the adversary has the necessary conditions for the attack to happen.

We define the origin of the attack as the `check-point-block` $\mathtt{B}_{\mathtt{chk}}$, which is the block at which a chain item occurs, or the destination block of the message containing the chain item. In the case of the alerting attack, the GMSC generates the chain item (*alerting pattern*) and hence the GMSC is our first $\mathtt{B}_{\mathtt{chk}}$. This is also shown in Node N of our attack graph (Figure 5). By performing the *forward dependency check* at each $\mathtt{B}_{\mathtt{chk}}$, the next chain item may be derived. Hence we perform the first *forward dependency check* at the GMSC. The forward dependency check consists of the two parts. Part 1 detects the next chain item(s) at the current $\mathtt{B}_{\mathtt{chk}}$, i.e. intra-block corruption (lines 3-10 in Figure 8). Part 2 detects the next $\mathtt{B}_{\mathtt{chk}}$ i.e., inter-block corruption (lines 10-13 in Figure 8).

Part 1 detects the next chain item(s) in the current $\mathtt{B}_{\mathtt{chk}}$ by using [IP rule 5]. Next chain item(s) in the current $\mathtt{B}_{\mathtt{chk}}$ may be detected in the three types of intra-block dependencies. The `seed` (*alerting pattern*) is assumed to be corrupt and stored in the current chain items array, l, contents of which are hence corrupt.

The current chain items array, l may be used in deriving the next corrupt chain item(s) in the current $\mathtt{B}_{\mathtt{chk}}$, under the following conditions: (1) if the current chain item(s) l is the input in a 'derivative dependency' (lines 3-4 in Figure 8) [1, 2 of IP rule 5]; (2) if the current chain item(s) l is used to corrupt block level data sources during a `Write`(), and the same items are read by another process and used as the input in a 'derivative dependency' (lines 5-6 in Figure 8) [3 of IP rule 5]; or (3) if the current chain items l are used to invoke other processes `Invoke`() and later used as the input in a 'derivative dependency' (lines 7-8 in Figure 8) [4 of IP rule 5]. As the current chain item (*alerting pattern*) does not derive the next chain items in the GMSC, we move on to the next part.

Part 2 detects the next $\mathtt{B}_{\mathtt{chk}}$ by using [5 of IP rule 5]. The occurrence of the next $\mathtt{B}_{\mathtt{chk}}$ may be detected by using inter-block dependencies (data items in signaling messages). The next $\mathtt{B}_{\mathtt{chk}}$ is the

block that receives corrupt data items in signaling messages from the current $\mathtt{B_{chk}}$ (lines 10-13 in Figure 8). The HLR receives the current chain item (*alerting pattern*) in the SRI message from the GMSC (current $\mathtt{B_{chk}}$). Hence the next $\mathtt{B_{chk}}$ is the HLR.

Using part 1 again, we find that the next chain item at the current $\mathtt{B_{chk}}$ (which is the HLR), but as the current chain item (*alerting pattern*) is not used to derive any other items in the HLR, we move on to part 2. The VLR is the next $\mathtt{B_{chk}}$ as it receives the current corrupt chain item (*alerting pattern*) in the PRN message from the HLR (current $\mathtt{B_{chk}}$). At the VLR, the *alerting pattern* i.e. current chain item, is the input to the OR derivative dependency whose output in the *page type*. Hence using condition 1 of part 1, we find the next chain item to be *page type*.

The forward dependency check is repeated until one of the following two *terminating conditions* are reached: (1) there are no outgoing messages from the checkpoint block with corrupt items i.e., next = Ø; or (2) there are no more messages to explore in the aCAT knowledge-base. In the case of the alerting attack, the forward dependency check stops when it is found that the BSS (current checkpoint block) does not generate any outgoing messages with corrupt items. During chain detection, due to telecommunication semantics, loops arise frequently and may be eliminated by checking looping conditions and ensuring that the same path is not traversed twice.

Input: Seed; **Output**: Chain[] = $\{c_0, c_1, c_2 \ldots c_n\}$, finite sequence of corrupt items; $\mathtt{c_0}$: Seed & $\mathtt{c_n}$: Goal;
chain_index: Array index of Chain[]
$\mathtt{B_{chk}}$: Check Point Block: Block containing chain item, or the destination block of the message carrying the chain item.
Let $l = \{l_0, l_1 \ldots l_k\}$ indicate the set of corrupt items in $\mathtt{B_{chk}}$.
Items in Chain and l are corrupt. Chain ⇒ corrupt(Chain) and l ⇒ corrupt(l)
Let next indicate the set of corrupt items in $\mathtt{B_{chk}}$ contained in the message propagating from $\mathtt{B_{chk}}$ to next block.
Procedure Forward dependency check(*seed*): [1]

1: Chain[0] ← *seed*; chain_index ← 0; Add *seed* to l
2: **repeat**
3: **for** (∀ $l_i \in l$) and (∀ Process in B_{chk}) **do**
4: **if** ($l_i \in \mathtt{fn^{Input}}$ and Dep_Operator) ⇒ corrupt($\mathtt{fn^{Output}}$) **then**
5: Chain[++chain_index] ← $\mathtt{fn^{Output}}$; Add $\mathtt{fn^{Output}}$ to l {. . . . . 1,2 of IP rule 5}
6: **else if** ($l_i \in$ {Write(), Read(), $\mathtt{fn^{Input}}$} and Dep_Operator ⇒ corrupt($\mathtt{fn^{Output}}$) **then**
7: Chain[++chain_index] ← $\mathtt{fn^{Output}}$; Add $\mathtt{fn^{Output}}$ to l {. . . . . 3 of IP rule 5}
8: **else if** ($l_i \in \{\mathtt{Invoke(P_x)}, \mathtt{fn^{Input}_{Px}}\}$ and Dep_Operator) ⇒ corrupt($\mathtt{fn^{Output}}$)] **then**
9: Chain[++chain_index] ← $\mathtt{fn^{Output}}$; Add $\mathtt{fn^{Output}}$ to l {. . . . . 4 of IP rule 5}
10: **end if**
11: **if** $\exists l_i \in \{d_a, d_b, \ldots\}$ in (message from $\mathtt{B_{chk}}$ to $\mathtt{B_j}$) **then**
12: next = $l_i \in \{d_a, d_b, \ldots\}$ in (message from $\mathtt{B_{chk}}$ to $\mathtt{B_j}$)
13: l = next; $\mathtt{B_{chk}} = \mathtt{B_j}$ {. . . . . 5 of IP rule 5}
14: **end if**
15: **end for**
16: **until** next ≠ Ø

Complexity: O(p) where p is number of processes in aCAT knowledge base.

Figure 8. Forward Algorithm

Trees are built as the next chain items are found. As we know, for seeds at every $\mathtt{B_{chk}}$ i.e. seed at every network location, an attack tree is built to show the propagation of corruption due to the seed.

[1] $\mathtt{fn^{Input}}$, Dep_Operator, $\mathtt{fn^{Output}}$, Read(), Write(), Invoke() . . . defined in Section 5

Hence in the alerting attack, for the seed (alerting pattern) at every B_{chk} a tree is built. In Figure 5, Tree 1 is built for seed at GMSC, Tree 2 is built for seed at message SRI and, Tree 3 is built for seed at message PRN.

In the third phase of the forward algorithm, trees are merged into graphs. All nodes at a level that are common to all trees are combined and then *pruned* to remove redundancy. The worst case performance of the algorithms may be approximated as O(p) where p is number of processes in aCAT knowledge base Table-2 and its size may be approximated to the number of messages in the service, whose vulnerability is subject to analysis. In the call delivery service, illustrated in Figure 1, there are 9 signaling messages, hence p is 9.

We also designed a reverse dependency check, similar to the forward dependency check, which also works in two parts. To find the previous chain items from the goal, the algorithm finds the previous B_{chk}'s.

Combinatory algorithms take as input multiple seeds or goals, and use the forward or reverse dependency checks to detect multiple chains of corrupt items as output. The worst case performance of these algorithms is O(Np), where N is the total number of chains. Although it is theoretically intriguing to study the damaging effects of *every* possible combination of seeds in a 3G network, in practice the adversary is only able to produce very few seeds in most cases. Therefore we developed the following practical optimizations when the number of seeds is relatively large:

Optimization 1. The combination of seed input sequence considered while building the tree is the order of input provided by the user i.e., the combinatory algorithm considers the first user input as the first corrupt item, the second user input as the second corrupt item and so on.

Optimization 2. Combinatory algorithms typically build a large number redundant of trees before pruning them. We reduce the number of redundant trees built by aggregating a number of seed locations to a single seed location.

6 Interesting Attacks discovered by aCAT

In this section, we present some interesting attacks detected by aCAT. aCAT constructs attack graphs assuming the worst case pre-conditions to be true. System administrators may use the attack graphs to perform more advanced network vulnerability analysis to check for example the probability of a certain attack.

The alerting attack and power-off power-on attack are discovered using only SDL knowledge. The call redirection attack presented in the Appendix is discovered with the help of expert knowledge.

6.1 Power-Off Power-On Attack

The attack graph for the power-off and power-on attack is shown in Figure 9. With the help of the guidelines defined in Section 3.2 the following may be derived.

Step 1: End User Effect: Goal nodes (Nodes A, B and C) identify end subscriber effects. Node A identifies that subscribers are disabled when they move to a new location as the previous location cancellation does not occur. Nodes B and C identify that incoming calls are not received by the subscriber as they may be sent to incorrect locations (Node B) or because subscriber page messages are sent to incorrect locations and hence not answered (Node C).

Step 2: Origin of Attack: Nodes at level 0 in Figure 9 indicate the target of the attack is the HLR.

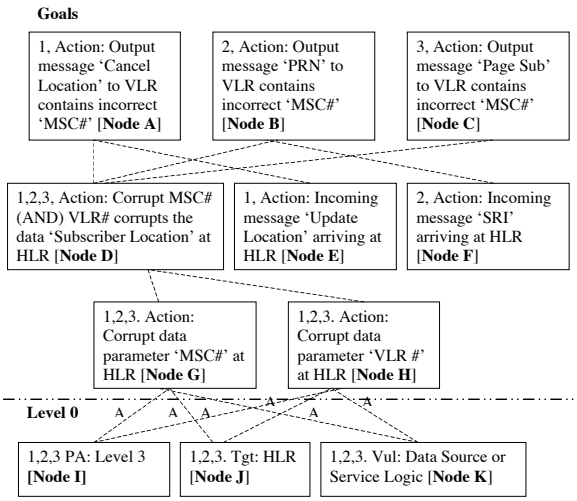

Figure 9: Attack Graph for Power Off Power On Attack

Step 3: Attack Propagation and Side effects: Nodes at other layers indicate the `seeds` used in the attack as the MSC number and VLR number. The MSC number and VLR number together indicate the current location of the subscriber. Hence corrupting the MSC number and VLR number corrupts the subscriber location. If the subscriber location is corrupt the network cannot locate the subscriber when a call arrives, resulting in resetting the location of the subscriber. The figure also illustrates the *AND derivative dependency* (Nodes G, H and D) and the *branching chain* i.e., both the data items 'MSC number' and 'VLR number' must be corrupt to corrupt data 'subscriber location' ($d_{MSC\ number}$ & $d_{VLR\ number} \rightarrow d_{subscriber\ location}$). Using these guidelines and general knowledge of the 3G network, the following attack scenario may be constructed.

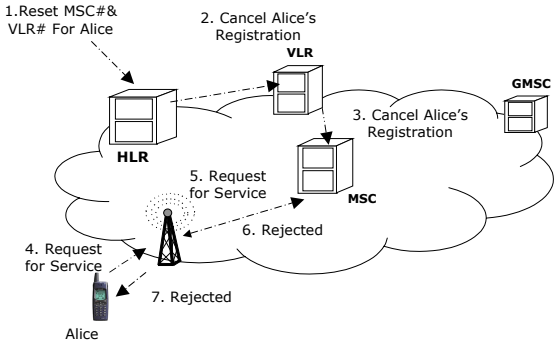

Figure 10: Power Off Power On Attack

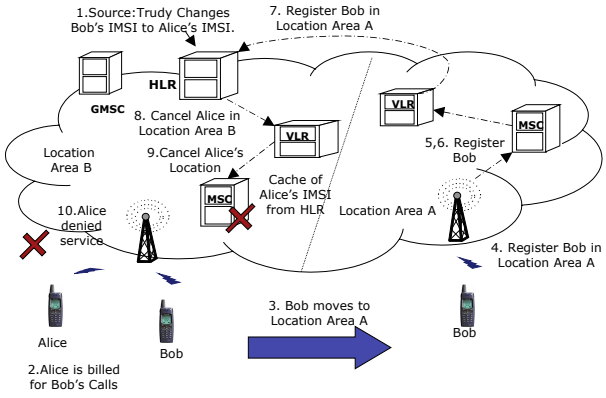

Figure 11: Mixed Identity Attack

Attack Scenario: This is an attack on the home network targeting the subscriber. This attack is illustrated in Figure 10. Trudy, the adversary, corrupts the 'MSC number' and 'VLR number' of Alice, the victim, to an unresolved value in the HLR. The HLR uses these parameters to locate a subscriber when a call request arrives. As the parameters are corrupt, the HLR cannot locate Alice when a call ar-

rives for her. The HLR thus resets Alice's 'MSC number' and 'VLR number' effectively de-registering her. After this, Alice cannot receive calls (Node 10). Alice will only begin to receive calls again if she power-cycles her phone, or if she changes location triggering a location update, thus refreshing the HLR with her correct MSC and VLR numbers. Note, that Trudy does not need to know valid MSC and VLR numbers for this attack to be effective.

6.2 Mixed Identity Attack

The attack graph for the mixed identity attack is shown in Figure 12. Using the guidelines in Section 3.2 attack scenario(s) may be derived as follows.

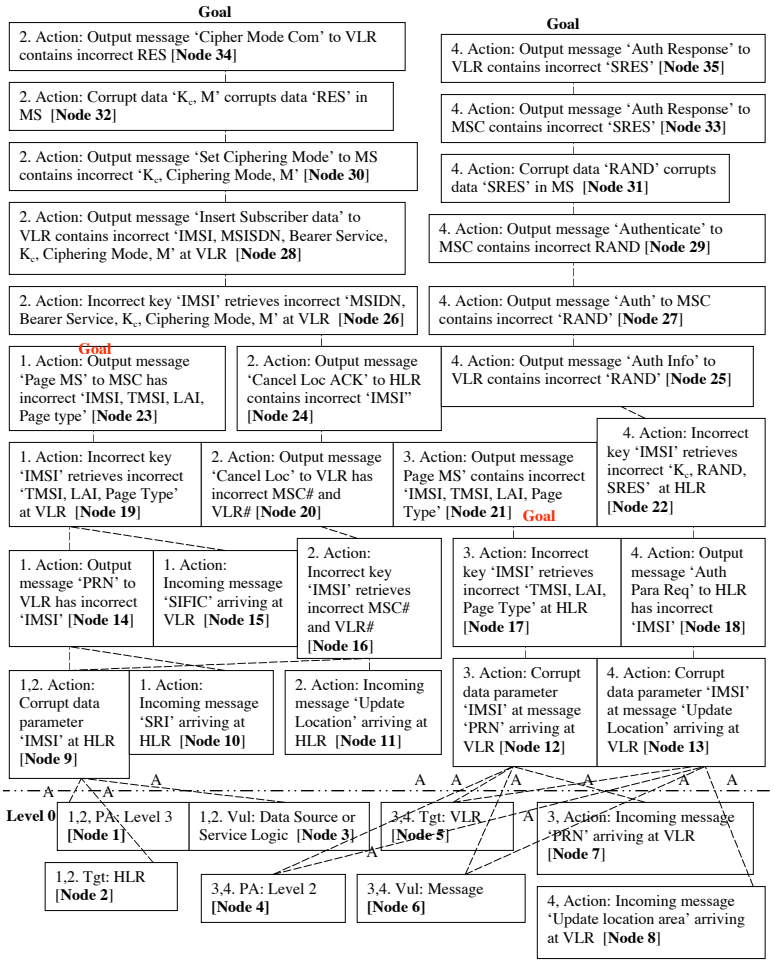

Figure 12: Attack Graph for Mixed Identity Attack

Step 1: End User Effect: Goal node(s) of Tree 1 and 3 (Nodes 23 and 21) identify the end effect as the inability of the subscriber to receive incoming calls due to incorrect paging. Goal node(s) of Trees 2 and 4 (Nodes 34 and 35) identify another effect of the attack as the inability of subscribers to authenticate themselves with the network.

Step 2: Origin of Attack: Nodes at level 0 indicate the origin of the attack. From Figure 12 it can be inferred that the two possible targets of the attack are the HLR or the VLR.

Step 3: Attack Propagation and Side effects: From other layers it can be inferred the `seed` used in the attack is the IMSI (Nodes 9, 12 and 13). The IMSI is the unique identity of a subscriber in the network. Corrupting the IMSI has a number of side effects as the IMSI is used as the key to retrieve incorrect parameters (Nodes 16, 17, 19 and 22). Corrupting the IMSI at the HLR or in signaling messages for incoming calls (Trees 1 and 3), leads to subscribers being paged incorrectly leading to loss of incoming calls. When a subscriber moves to a new location, a corrupt IMSI at the HLR leads to registration cancelation of the incorrect subscriber in the old location (Tree 2, Node 20) and failed authentication of the subscriber at the new location (Tree 2, Node 34). The linear chain is demonstrated by Tree 2. Using these results and general knowledge of the 3G network, the following attack scenarios may be constructed.

Attack Scenario 1: This is an attack on the home network targeting the network itself. In this attack, the identities of a group of victims are mapped to a designated victim's identity. The designated victim is charged for all the calls. This attack is illustrated in Figure 11. Trudy maps the IMSI of all victims to that of Alice at the HLR. The VLR of location area B has uncorrupt data and hence all the mobile stations will be enabled. When Alice moves to a new location, she requests the VLR in the new location area A to register her. The VLR in the new location area A registers Alice and requests the HLR to do the same. The HLR registers Alice and cancels Alice's previous location i.e., the current location of rest of the victim group. This results in cancellation of all non-traveler victims. Alice, the designated victim is charged for all victim group's calls before the cancellation; after the cancellation, none of the non-traveller victims can receive calls.

Attack Scenario 2: Trudy replaces the IMSI of organization Alice Inc. (victim) in every incoming call, with IMSI of rival organization (Bob Inc.) (Tree 3, Node 12). Every call arriving at the network for Alice Inc. is received by Bob Inc.

6.3 Other Attacks

Due to space limitations we summarize two other interesting attacks found by aCAT. The *missed call attack* shows that by capturing 'update location' messages on the air interface and modifying a single parameter it is possible to completely deny registration for all subscribers arriving at a location. The *call redirection attack* shows that by modifying a single parameter it is possible to completely redirect all calls arriving at a location. For detailed information on these attacks please refer to the Appendix. In the next section, we describe the algorithm used in finding these attacks.

7 Experimental Results

We conducted experiments to measure the utility of including expert input into the aCAT knowledge base. Utility of expert input is measured by comparing the percentage of attacks detected using SDL knowledge with expert input and by using SDL knowledge only.

In a realistic setting every service node provides a number of services. To simulate this setting, in our experiments, we considered the call origination, location registration, and location updating services in-addition to the call delivery service. We obtained the SDL knowledge from the specifications, for call handling [3], and for mobility [4]. We formatted this knowledge with the network dependency model and also some expert input.

All the above services combined together constitute 46 signaling messages in total and 46 processes to handle these signaling messages. Hence each table in our aCAT knowledge base comprises of 46

rows. We tested this aCAT knowledge base (which contains combined data of call origination, location registration, location updating, and call delivery services) for single and double seed combinations in the forward and reverse algorithms (illustrated in Table 3 and 4).

When the user input corresponded to areas where SDL knowledge is adequate, we found that expert knowledge does not show any improvement in number of attacks detected. On an average, goals detected by SDL knowledge cover 70% of all the possible goals that are detected with expert input. The seeds detected by SDL knowledge cover 77% of all of the possible seeds that are identified with expert input.

User Input: # of Seeds	SDL and Expert Knowledge		SDL only Data		% Extra goals Detected
	# of Nodes	# of Goals /# of Paths	# of Nodes	# of Goals /# of Paths	
1	26	6	14	3	50%
	35	5	30	2	60%
	48	7	32	4	42%
2	77	10	69	9	10%
	89	11	54	7	36%
	103	16	83	10	37%

Table 3: Forward Experimental Results

User Input: # of Goals	SDL and Expert Knowledge		SDL only Data		% Extra Seeds Detected
	# of Nodes	# of Goals /# of Paths	# of Nodes	# of Goals /# of Paths	
1	23	6	16	4	33%
	32	5	28	5	0%
	45	7	36	6	14%
2	77	10	71	4	60%
	86	11	65	9	18%
	103	16	103	16	0%

Table 4: Reverse Experimental Results

In conducting the above experiments, we also observed that in some cases, corrupting a single data item can effect not one but multiple services. Our attack graphs presented in Section 6 and Appendix are a testament to this fact. For example, in the mixed identity attack, the adversary corrupts the data item 'IMSI' (Node 9 in Figure 12) which effects the delivery of calls in the call delivery service (Node 23 in Figure 12), subscriber authentication in location registration service (Node 34 in Figure 12), and subscriber authentication in location updating service (Node 35 in Figure 12).

In effect, the aCAT knowledge base need not be restricted to the above four services. Any other services (such as SMS) with specifications written in SDL language may be added to the aCAT knowledge base by application of our network dependency model alone.

The results of these experiments indicate the usefulness of SDL knowledge for both aCAT and adversaries. For aCAT, SDL provides a significant portion of the knowledge base. For adversaries, SDL provides a rich set of possible attacks. In fact, it provides information that may be used to launch the large majority of possible attacks. Furthermore, the remaining attacks are still possible even without the direct targeting of an adversary.

8 Discussion

8.1 Limitations in SDL

Although SDL specifications are free, easy to obtain and contribute to 70% of the total network knowledge in aCAT. In conducting experiments detailed in the previous section, we identified inadequacies in SDL. We detail how aCAT alleviates these problems and we prove that aCAT can be a important tool in uncovering sophisticated cascading attacks in advance.

SDL is inadequate in certain syntactic, semantic and technical areas. In the *syntactic and semantic areas*, SDL specifications have: (1) *limited expressiveness*; and make (2) *in-explicit assumptions*. Due to these limitations, SDL is inadequate in defining functions (for e.g., function *Derive GSM BC from*

ISDN compatibility) in process resulting in inaccurate cascading effect detection. aCAT overcomes this problem by formating SDL using the network dependency model and detect attacks using infection propagation rules.

In technical areas, SDL is lacking in representing *authentication*, and *location* functions. In the *authentication* area, SDL is in-explicit with the data items used and their derivative dependencies, resulting in inaccurate attack detection. aCAT alleviates this problem by the usage of expert knowledge.

With respect to *location data*, SDL does not take into consideration the location data items used to route signaling messages. Corruption of these items results in incorrect routing of signaling messages. aCAT alleviates this problem by hardcoding the necessary details.

Adversaries may also alleviate the inadequacies in SDL, and devise much more devastating attacks, by careful in-depth study of the specifications or by gaining unauthorized access to specific requirements documents. We would also like to state that attacks can still be devised even without this extra knowledge, even if some of the sophisticated cascading effects are unanticipated by the adversary.

8.2 Defenses

So far, we have discussed how aCAT can successfully detect cascading attacks. We now discuss how the 3G network can defend against such attacks. We would like to point out that defending against cascading attacks is not an easy task. This is because, in cascading attacks, corrupt values correspond to system acceptable values. Hence just error checking is insufficient to detect corruption, which may lead to a cascading attack. Error checking must also comprise of context checking every time a data item is generated. An example of such context-based error checking in the alerting attack is, checking the cause for creation of the *alerting pattern* data item. The cause is the arrival of the `IAM` signaling message. The `IAM` message is an indication of a CIRCUIT SWITCHED CALL, hence if *alerting pattern* takes on any other value other than CIRCUIT SWITCHED CALL, it is an error and corruption is detected.

Such solutions may be expensive to implement as they must check context of each and every data item and may slow down the network drastically. Even so, such context-based error checking solutions may not be successful all the time, because such solutions may only be able to detect corruption at the origin and not after it cascades to some remote service node. Hence, such context checking solutions must be implemented at each and ever service node by every service provider. We believe this approach is very promising. Context-based error checking and attack detection will be explored as a part of our future work.

9 Related Work

Our related work covers *3G network vulnerability assessment* and *attack-graph technologies*.

3G network vulnerability assessment: Telecommunication specifications [1, 2, 5] specify 3G security and identify security threats. Howard et al. [17], El-Fishway et al. [14], Lo et al. [23], Welch et al. [41] and, Clissmann et al. [11] identify attack scenarios on 3G networks while trying to prove the inadequacy of current security schemes. They also present new architectures for 3G security. Mitchell et al. [24], Boman et al. [9] and Bharghavan et al. [8] discuss the security features available in current 3G networks. Brookson [10] motivates the need for security. Kotapati et al [21] present a taxonomy of cyber attacks in 3G networks. *However, they neither use open specifications to devise cascading attacks nor perform attack-graph based vulnerability analysis.*

CAT [20] is the first attempt to apply the Internet attack graph technology to analyze the vulnerabilities of 3G networks. Although CAT also uses SDL specifications, aCAT is significantly different from CAT because aCAT incorporates: (1) the unique network dependency model and infection propagation rules for accurate chain detection and; (2) expert knowledge for exhaustively uncovering all possible cascading attacks.

Attack-graph technologies: In the Internet domain attack-graph technologies have been extensively studied by [7, 12, 13, 18, 19, 27–37, 40, 42]. However these technologies are not designed to handle 3G semantics i.e. dependencies, and infection propagation rules. The earlier work in this area includes Netkuang [42], a network configuration vulnerability checker which performs a goal based breath first search. Swiler and Philips [31, 36, 37] generate their attack graphs by backward exploration from the goal given atomic attacks as input. In contrast, our forward algorithm generates attack graphs by forward exploration from seeds to goals.

Model checking is a major technique proposed for automatic attack graph generation and has been used by Ritchey and Amman [33] for vulnerability analysis of a network, Sheyner et al. [34] for automatic generation of attack graphs, and Ramakrishnan and Sekar [32] for identifying configuration vulnerabilities. However, all the above techniques have the disadvantage of not being scalable. Ammann et al. proposed a fix to this problem by exploiting a monotonic assumption to achieve scalability [7]. In [30], Ou et al. proposed a logic-programming method for scalable network vulnerability analysis. Jha et al. [18, 19, 35] have analyzed attack graphs in terms of properties such as survivability, reliability, etc. They also present a minimization technique that allows analysts to decide which minimal set of security measures guarantee the safety of the system.

The semantics of aCAT attack graphs are fairly different from Internet attack graphs. For example, in Internet attack graphs only exploits can cause state transitions, while in aCAT attack graphs many state transitions are caused by legitimate actions. Nevertheless, by viewing the infection propagation actions of aCAT as accidental "unintentional exploits", the attack graphs generated by aCAT could be conceptually deduced to an Internet attack graph [7, 34]. Accordingly, existing attack graph analysis techniques (e.g., [18, 19, 35]) could be applied to analyze the graphs generated by aCAT. In this sense, existing attack graph analysis techniques are complementary to aCAT. Nevertheless, it should be noticed that the focus of this paper is to generate and interpret 3G specific attack graphs, although analyzing them is part of our future work.

From the viewpoint of attack graph generation, please note that besides the new cascading attacks identified, the main contribution of this paper is the 3G specific network dependency model and infection propagation rules, which are necessary for any attack graph generation method (e.g., model checking) to work in 3G networks to identify cascading attacks. We propose a light-weight, ad-hoc attack graph generation algorithm instead of adopting an existing attack graph generation method such as model checking and [7] since sometimes a 3G network administrator may not be able to pre-determine completely or precisely which types of goal nodes satisfy the attacker's goal and which do not, while both model checking and [7] require the goal states be explicitly pre-determined (during vulnerability analysis).

In the sense of exploiting certain network dependencies to "chain" attack actions together, our work is relevant to alert correlation research (i.e., [12, 13, 27–29, 40]). Nevertheless, alert correlation is an intrusion detection activity that needs alerts to be raised in prior, while aCAT does not need any alerts.

10 Conclusion

In this paper, we presented a unique solution **aCAT** for detecting new cascading attacks on the 3G network. Cascading attacks pose a great threat to the 3G network as they are extremely subtle and can have many far reaching remote effects. aCAT has many applications, it may be used for real-time detection of attack origin, and for detection of vulnerable areas in the network. aCAT could be mandated for use, by wireless telecommunication policy makers, to detect sources of unusual network events and to protect the network from outage due to cascading attacks, during critical events such as 9/11. aCAT also be used to check (hence trace corruption) incoming traffic from certain rogue networks. Our future plans include extending aCAT to include feasibility analysis, proposal of defense mechanisms, and automating the process of deriving attack scenarios from attack graphs. Readers interested in additional cascading attacks please refer to the Appendix.

References

[1] 3GPP. 3g security; security principles and objectives. Technical Standard 3G TS 33.120 V3.0.0, 3G Partnership Project, May 1999.

[2] 3GPP. 3g security; security threats and requirements. Technical Standard 3G TS 21.133 V3.1.0, 3G Partnership Project, Dec. 1999.

[3] 3GPP. Basic call handling - technical realisation. Technical Standard 3GPP TS 23.018 V3.4.0, 3G Partnership Project, April 1999.

[4] 3GPP. Mobile application part (map) specification. Technical Standard 3GPPTS 29.002 V3.4.0, 3G Partnership Project, April 1999.

[5] 3GPP. A guide to 3rd generation security. Technical Standard 3GPP TR 33.900 V1.2.0, 3G Partnership Project, Jan. 2001.

[6] G. 3GPP. Third Generation Partnership Project. In *http://www.3gpp.org/*.

[7] P. Ammann, D. Wijesekera, and S. Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM conference on Computer and communications security (CCS '02)*, pages 217–224, November 2002.

[8] V. Bharghavan and C. Ramamoorthy. Security Issues in Mobile Communications. In *Proceedings ISADS 95. Second International Symposium on Autonomous Decentralized Systems*, pages 19–24, April 1995.

[9] K. Boman, G. Horn, P. Howard, and V. Niemi. Umts security. *Electronics Communications Engineering Journal: Special issue security for mobility*, 14(5):191–204, October 2002.

[10] C. B. Brookson. Security in current systems. In *IEE Colloquium on Security in Networks*, number Digest No. 1995024, pages 3/1–3/6, February 1995.

[11] C. Clissmann and A. Patel. Security for mobile users of telecommunication services. In *Universal Personal Communications*, ICUPC '94, pages 350–353, October 1994.

[12] F. Cuppens and A. Miège. Alert correlation in a cooperative intrusion detection framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 202–215, May 2002.

[13] F. Cuppens and R. Ortalo. Lambda: A language to model a database for detection of attacks. In *Recent Advances in Intrusion Detection*, pages 197–216, 2000.

[14] N. A. El-Fishway, M. A. Nofal, and A. M. Tadros. An Improvement on Secure Communication in PCS. In *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, pages 175–182, April 2003.

[15] J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL, Formal Object-oriented Language for Communicating Systems*. Prentice Hall, 1997.

[16] W. Enck, P. Traynor, P. McDaniel, and T. F. La Porta. Exploiting open functionality in sms-capable cellular networks. In *CCS '05: Proceedings of the 12th ACM conference on Computer and communications security*. ACM Press, 2005.

[17] P. Howard, M. Walker, and T. Wright. Towards a coherent approach to third generation system security. In *Second International Conference, 3G Mobile Communication Technologies*, pages 21–27, Nov 2001.

[18] S. Jha, O. Sheyner, and J. Wing. Two formal analys s of attack graphs. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, page 49, Washington, DC, USA, June 2002. IEEE Computer Society.

[19] S. Jha, O. Sheyner, and J. M. Wing. Minimization and reliability analyses of attack graphs. Technical Report CMU-CS-02-109, February 2002.

[20] K. Kotapati, P. Liu, and T. F. La Porta. CAT – A Practical Graph & SDL Based Toolkit for Vulnerability Assessment of 3G Networks. In *Proceedings of the 21st IFIP TC-11 International Information Security Conference, "Security and Privacy in Dynamic Environments"*, SEC 2006, May 2006.

[21] K. Kotapati, P. Liu, Y. Sun, and T. F. La Porta. A Taxonomy of Cyber Attacks on 3G Networks. In *Proceedings IEEE International Conference on Intelligence and Security Informatics, ISI*, Lecture Notes in Computer Science, pages 631–633. Springer-Verlag, May 2005.

[22] C. Lee, M. Hwang, and W. Yang. Enhanced privacy and authentication for the global system for mobile communications. *Wirel. Netw.*, 5(4):231–243, 1999.

[23] C. C. Lo and Y. J. Chen. Secure communication mechanisms for GSM networks. In *IEEE Transactions on Consumer Electronics*, Lecture Notes in Computer Science, pages 1074–1080, Nov 1999.

[24] C. Mitchell. Security techniques. *Proceedings of the IEE Electronics Division Colloquium on Security in Networks*, 14(IEE (London) Digest No: 1995/024):2/1–2/6, February 1995.

[25] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. *IEEE Security and Privacy*, 1(4):33–39, 2003.

[26] T. Moore, T. Kosloff, J.Keller, G. Manes, and S. Shenoi. Signaling System 7 (SS7) Network Security. In *Proceedings of the IEEE 45th Midwest Symposium on Circuits and Systems*, August 2002.

[27] P. Ning, Y. Cui, and D. S. Reeves. Constructing Attack Scenarios through Correlation of Intrusion Alerts. In *Proceedings of the 9th ACM Conference on Computer & Communications Security (CCS '02)*, pages 245–254, Nov 2002.

[28] P. Ning and D. Xu. Learning Attack Strategies from Intrusion Alerts. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03)*, pages 200–209, Oct 2003.

[29] P. Ning, D. Xu, C. G. Healey, and R. A. St. Amant. Building Attack Scenarios through Integration of Complementary Alert Correlation Methods. In *Proceedings of the 11th Annual Network and Distributed System Security Symposium (NDSS '04)*, pages 97–111, Feb 2004.

[30] X. Ou, S. Govindavajhala, and A. Appel. MulVAL: A Logic-based Network Security Analyzer. In *Proceedings of the 14th Usenix Security Symposium*, pages 113–128, August 2005.

[31] C. Phillips and L. P. Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms (NSPW '98)*, 1998.

[32] C. R. Ramakrishnan and R. C. Sekar. Model-Based Analysis of Configuration Vulnerabilities. *Journal of Computer Security*, 2002.

[33] R. W. Ritchey and P. Ammann. Using model checking to analyze network vulnerabilities. In *Proceedings 2000 IEEE Computer Society Symposium on Security and Privacy*, volume 00, pages 156–165, Los Alamitos, CA, USA, May 2000. IEEE Computer Society.

[34] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, May 2002.

[35] O. Sheyner and J. Wing. Tools for Generating and Analyzing Attack Graphs. In *Proceedings of Formal Methods for Components and Objects*, Lecture Notes in Computer Science, pages 344–371, 2005.

[36] L. Swiler, C. Phillips, D. Ellis, and S. Chakerian. Computer-Attack Graph Generation Tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II*, June 2001.

[37] L. P. Swiler, C. Philips, and T. Gaylor. A Graph-Based Network Vulnerability Analysis System. SandiaReport SAND97-3010/1, Sandia National Laboratories, January 1998.

[38] Switch. 5ESS Switch. In *http://www.alleged.com/telephone/5ESS/*.

[39] Telcoman. CENTRAL OFFICES. In *http://www.thecentraloffice.com/*.

[40] S. J. Templeton and K. Levitt. A requires/provides model for computer attacks. In *Proceedings of the 2000 workshop on New security paradigms (NSPW '00)*, pages 31–38, New York, NY, USA, September 2000. ACM Press.

[41] D. Welch and S. Lathrop. Wireless Security Threat Taxonomy. In *IEEE Workshop on Information Assurance*, IEEE Systems, Man and Cybernetics Society Information Assurance Workshop, pages 76–83, Jun 2003.

[42] D. Zerkle and K. Levitt. NetKuang–A multi-host configuration vulnerability checker. In *Proceedings of the Sixth USENIX Security Symposium*, pages 195–201, 1996.

Appendix: Interesting Attacks

In this section, we present several other interesting attacks discovered by aCAT. These attacks are detailed below.

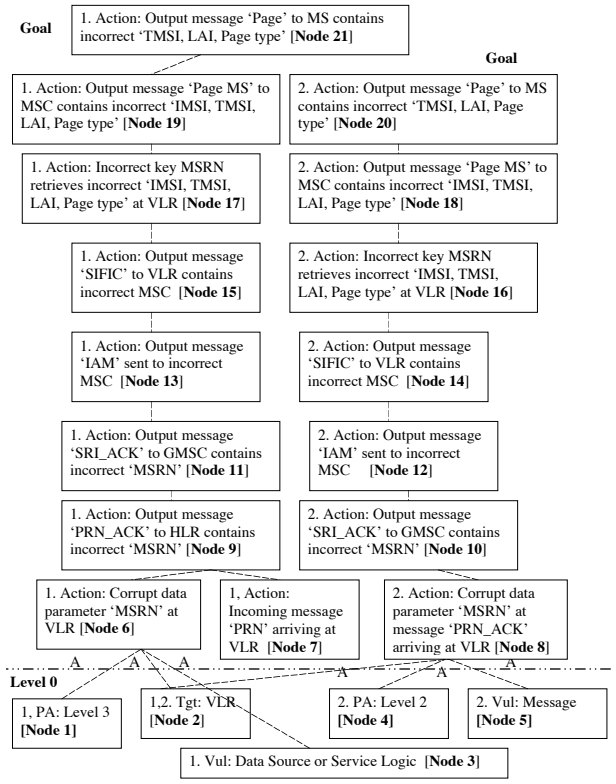

Figure A1: Attack Graph for Call Redirection Attack

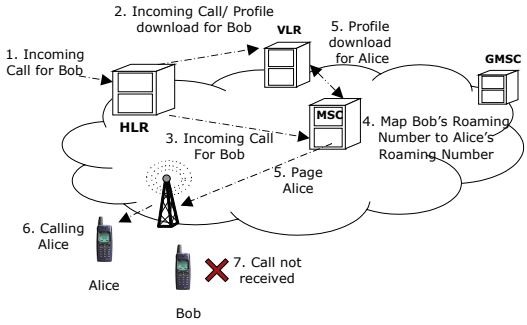

Figure A2: Call Redirection Attack

A.1 Call Redirection Attack

The attack graph for the call redirection attack is shown in Figure A1. Using the guidelines in Section 3.2 attack scenario(s) may be derived as follows.

Step 1: End User Effect: Goal nodes (Nodes 21 and 20) identify the end result of the attack as redirection of incoming calls to other subscribers. This is because the 'Page message' is routed to the incorrect subscriber (due to incorrect `goal` items TMSI and LAI).

Step 2: Origin of Attack: Nodes at level 0 in Figure A1 indicate that the target of the attack is the VLR.

Step 3: Attack Propagation and Side effects: Nodes at the other layers indicate the `seed` used in the attack is the MSRN (roaming number). The MSRN is used to route incoming calls to the designated subscriber. Each incoming call is assigned a MSRN. By switching the MSRNs (Nodes 10 and 11) it is possible to switch the destination of the incoming call (Nodes 12 and 13) and send the call to the incorrect subscriber (Nodes 20 and 21). Tree 1 and 2 are examples of the line cascading effect.

Using the above guidelines and general knowledge of the 3G network, the following attack scenario may be constructed.

Attack Scenario 1: This is an attack on the visiting network targeting the network itself. In this attack, the roaming number of subscriber Bob is corrupt (possibly by incrementing or decrementing the number). This results in the redirection of the Bob's calls to another subscriber Alice (to whom the corrupt roaming number has been assigned). Hence Alice is alerted of the incoming call instead of Bob. Another side effect of corrupting the MSRN is that if the MSRN is not assigned to any other subscriber the call is dropped. For every incoming call arriving at the MSC for Bob, the adversary changes the roaming number to an unassigned value, in the incoming call. Hence Bob is not alerted of the incoming call. Bob can make calls and update his profile but can never receive his calls at a particular location and is unaware of this problem. This problem may be rectified if Bob travels to a new location. This attack is illustrated in Figure A2.

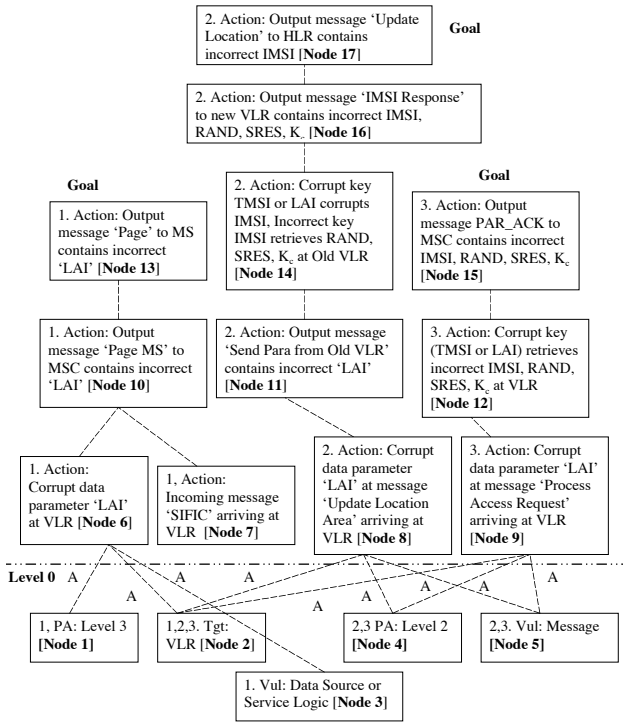

Figure A3: Attack Graph for Missed Calls

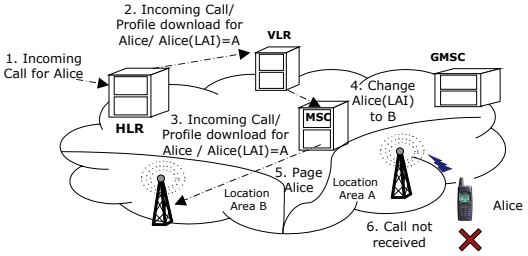

Figure A4: Missed Calls Attack

A.2 Missed Calls Attack

The attack graph for the missed calls attack is shown in Figure A3. With the help of the guidelines defined in Section 3.2 attack scenarios may be derived as follows.

Step 1: End User Effect: The goal node (Node 13) identifies that incoming calls are not received by the subscriber as Page messages are sent to incorrect location (incorrect LAI). Goal node (Node 17) identifies that subscribers are disabled when they move to a new location because updates due to location changes (Update location message) contain incorrect subscriber identity (incorrect IMSI) resulting in cancelation of the update.

Step 2: Origin of Attack: Nodes at level 0 in Figure A3 indicate that the target of the attack is the VLR.

Step 3: Attack Propagation and Side effects: Nodes at the other layers indicate the seed used in the attack is the LAI (Location Area Identifier). The LAI identifies the location the subscriber is currently visiting. The combination of the TMSI and LAI uniquely identifies a subscriber and their current location i.e., at the VLR. Corruption of either the TMSI or the LAI results in corruption of subscribers identity (Node 14). Node 14 is an illustration of an OR node.

Using the above guidelines and general knowledge of the 3G network, the following attack scenarios may be constructed.

Attack Scenario 1: This is an attack on the visiting network targeting the subscriber. In this attack, the location area (LAI) of the victim is corrupt during the incoming call profile download. The corrupt LAI causes the subscriber to be paged at the incorrect location for incoming calls and results in the subscriber not receiving the call. Subscribers can make calls and update their profile but can never receive incoming calls at this particular location. Receiving calls if moved to another location is possible. This attack is illustrated in Figure A4.

Attack Scenario 2: This is another attack scenario derived from the Tree 2 in the attack graph shown in Figure A3. This is an attack on the visiting network targeting the network itself. The victim in this attack is Wireless XYZ Inc. (the company providing service in a location area A). Subscribers currently visiting this area A, cannot register with the network. This because the subscriber's LAI in the update location message is corrupt, resulting in retrieval of incorrect IMSI and authentication material (Node 14). The subscriber responds incorrectly to the incorrect authentication material and as a result cannot register with the network. An adversary may capture all update location messages on the air interface and corrupt the LAI's thereby preventing subscriber registration.