Graph based Metrics for Intrusion Response Measures in Computer Networks [1]

Marko Jahnke, Christian Thul
Research Establishment for Applied Science (FGAN)
Research Institute for Communication,
Information Processing and Ergonomics (FKIE)
Wachtberg, Germany
Email: {jahnke|thul}@fgan.de

Peter Martini
University of Bonn, Institute of Computer Science IV
Bonn, Germany
Email: martini@cs.uni-bonn.de

***Abstract*—This contribution presents a graph based approach for modelling the effects of both attacks against computer networks and response measures as reactions against the attacks. Certain properties of the model graphs are utilized to quantify different response metrics which are well-kown from the pragmatic view of network security officers.**

Using these metrics, it is possible to (1) quantify practically relevant properties of a response measure after its application, and (2) estimate these properties for all available response measures prior to their application. The latter case is the basis for the selection of an appropriate reaction to a given attack.

Our graph-based model is similar to those used in software reliability analysis and was designed for a scalable granularity in representing properties of the network and its components to be protected. Different examples show the applicability of the model and the resulting metric values.

I. Introduction

Attacks against computer systems and networks in their different characteristics are omnipresent and thus not surprising anymore. Almost every network that connects computers has been facing processes of reconnaissance, penetration, stealing or damaging information in the past, with more or less serious subsequent effects.

When an attack has been indicated by a monitoring system, network security officers need to select an appropriate response to the attack carefully. The way how to define this 'appropriateness' heavily depends on the properties and the deployment objective of the network and its components. There is only a small number of approaches selecting response mechanisms automatically; this is mainly caused by too many possibilities to damage a system rather than mitigating the effects of an attack.

This contribution proposes an approach for modelling the systems to be secured and the effects of attacks and responses using directed graphs, comparable to data structures used in software reliabiliy analysis. Quantified properties of these graphs are used as metrics for assessing the value of a response mechanism. These metrics are aligned to the pragmatic view of network security officers, and they include the original deployment objective of the network.

[1]Published in: Proc. of the 3rd IEEE Workshop on Network Security, held in conjunction with 32th IEEE Conference on Local Computer Networks, Dublin, Oct. 2007.

II. Related Work

The careful selection of reaction mechanisms to attacks against computer networks has always been a challenging field of work. Different contributions have been dealing with cost models in the area of intrusion detection and response. A general overview on existing work in the area of intrusion response has been published by Stakhanova et al. in [SBW07].

In her book [Den99], Denning stated that cost analysis in the area of IT security – and risk analysis in general – simply cannot be seen as an exact science, since in many cases, relevant values cannot be quantified at all. Northcutt describes in [Nor99] the (informal) process of risk analysis in IT systems and defines the value of resources by their *criticality* as well as their *lethality*. These values are assigned in an increasing order to, e.g., Windows Workstations, UNIX Workstations, Web Servers, DNS Servers up to Firewalls/Paketfilters and Routers. Parts of this approach have been included during the design process of the model described in this paper.

Lee et al. [LFM$^+$02] identified different operational costs as metrics for selecting intrusion response measures. Starting with a taxonomy of attacks that have been given by a reference dataset, empirical costs for the attack *damage* and *reactions* have been defined. The damage of a resource is defined as product of its criticality and its (estimated) complexity of the reaction. The main focus of this work is more on reducing costs for the security event detection.

Toth and Kruegel [TC02] looked at the effects of a reaction in a network model that considers resources (applications/services) as well as users, the network topology as well as access control (firewall rules). In general, for all mentioned model components, a *capability* value is defined. Also, the respective inter-dependencies of resources have been modeled; so called 'dependency trees' express these relationships. By the increase of resource availability, the value of response measures is estimated. Our approach extends this idea by using general directed graphs with different kinds of dependencies between resources and by deriving quantitative differences between system states from these graphs. Other approaches are also relying on resource graphs, but for different purposes:

Balepin et al. [BMR$^+$03] extended the idea of representing services and their inter-dependencies in a graph for selecting responses through creating a resource type hierarchy, so that every service type has common response measures associated with it. Response sequences need to be optimal for each service node, i.e every response step needs to produce maximum benefit at minimum costs.

ADEPTS [WFB$^+$07] is a complex framework for determining automated responses against attacks, based on two types of graphs: a service graph that expresses inter-dependencies between available services, and an attack graph that represents possible attack states and their probabilities. Responses are selected based on their effectiveness during previous applications in the past. This framework has been successfully applied and evaluated in an E-Commerce application context. However, we believe that our approach for determining the value of a countermeasure reflects the dynamics of the situational context more accurate.

Our model has a lot of similarities to software reliability analysis methodologies, such as the 'Scenario-Based Reliability Estimation' (SBRE, see e.g. [YCA04], [YA02]), where comparable data structures and algorithms are deployed. Our approach basically broadens the application area for these techniques to the networking context.

III. The Pragmatic View: How Security Officers choose Responses

Network security officers (NSOs) are usually equipped with more or less complex diagnosis systems and applications, such as network management systems (NMS), intrusion detection systems (IDS), intrusion prevention systems (IPS) and additional tools like administrator consoles.

Conventionally, an NSO picks a selection of the available response measures together with the appropriate parameters and triggers it manually, at the console of the penetrated systems or even remotely over the network. When choosing the response measures and their parameters, NSOs often take the following factors into account:

- *Expected Response Success*
  Clearly, the most important aspect is the expected success of a measure. Negative side effects (e.g. unwanted partial inavailability) need to be considered here. As long as a reaction does not likely have a positive effect (whatever this means in the according application scenario) on the network, it will not be chosen. This also holds for the response parameters.
- *Expected Response Effort*
  Maybe the second most important aspect is the estimated effort (or costs) that is needed for performing response measures. If two sets of possible responses have the same expected success, most probably the set will be selected, which is easier to apply.
- *Expected Response Error-Proneness*
  The (subjective) probability of failing when performing a response measure is also very important. It cannot ultimately be precluded that a wrong selection of response measures and their parameters will put the monitored system into a state worse than caused by the attack itself. So, in most cases, the complication-less alternative would be selected by an NSO.
- *Expected Response Durability*
  The expected duration of the response effects is probably an aspect that is less important than the other three mentioned above. If two alternative sets of responses promise comparable values for the other aspects, most likely the one with the longer expected durability will be chosen, i.e. the expected time period after which additional actions will get necessary for keeping the system healthy.

Of course, there are more aspects to be considered by NSOs, but these strongly depend on the corresponding deployment scenario.

If connected with access control enforcement points (e.g. firewalls, packet filters), some up-to-date IDS/IPS solutions allow automatic or semi-automatic selection of response mechanisms when certain attacks are detected. Examples include TCP session termination as soon as suspicious packet payloads are detected in the monitored protocol stream.

In many cases (e.g. Snort Inline [Sno07]), the reaction itself is coded in the detection signature that has been specified prior to the deployment of the system. Thus, this can simply be viewed as a suggestion of the signature writer. However, in these cases, there is no dynamic on-line estimation of the response involved. In high-assurance environments, this static approach of selecting response measures is obviously not appropriate.

IV. The Graph Model

In this section, the graph based model for estimating response metrics is discussed and some results for example scenarios are presented. Here, we restrict ourselves to first discuss *availability* as one important and representative objective of IT security. In section V, a potential way of extending the aproach to e.g., confidentiality, integrity, and authenticity, is proposed.

A. Resources, and Availailabilities

As already suggested by Toth and Kruegel [TC02], our model is based on properties of *resources*. The set of resources is furtheron denoted as $\mathcal{R}$. Resources can either be *service instances* (instances of a service provided by hardware, operating system, applications or network services) or *users*. The respective sets are furtheron denoted as $\mathcal{S}$ and $\mathcal{U}$ with $\mathcal{R} = \mathcal{S} \cup \mathcal{U}$ and $\mathcal{S} \cap \mathcal{U} = \emptyset$

Concerning availability, we observe different kinds of dependencies between resources. The users depend on applications and services within the network to conduct a certain mission – otherwise the network would be completely useless for them. On the other hand, applications often rely on other applications and services, such as many network communication systems are depending on the availability of directory services and of the network transport service itself.

We assume that every resource $r \in \mathcal{R}$ of a system to be secured has a certain *availability*, expressed as a value $A(r) \in$

$[0,1]$ E.g. if a router is able to handle only 10% of the traffic it was designed for, its current availability is denoted as 0.1. There intuitively is a lower bound for totally inoperable service instances and an upper bound for instances which operate with full capabilities (i.e. operate fully as designed).

A current availability value of a resource can be both inherent, if it is indicated by a diagnosis system (NMS, IDS/IPS), and a result of the propagation of changed availabilities of other resources the currently considered resource depends on. Thus, we assign an *intrinsic* availability value $A_I(r) \in [0,1]$ and a *propagated* availability value $A_P(r) \in [0,1]$, so that they are statistically independent from each other. We define the resulting availability as

$$A(r) = A_I(r) \cdot A_P(r) \tag{1}$$

to every resource $r \in \mathcal{R}$. The former one expresses the availability of the instances inner functionalities and the latter expresses a combination of the availabilities of the resources r depends on.

For an edge $(r,s) \in E$, a value $A(s) > 0$ indicates that r has *direct accessibility* to s (i.e. without other resources as mediators). If $(r,s) \notin E$ or $A(s) = 0$, there is no direct access possible from r to s. If there is a path $(r_1,r_2),\ldots,(r_{n-1},r_n)$ with $(r_i,r_{i+1}) \in E \wedge A(r_i) > 0 \;\forall r_1 \neq \ldots \neq r_n \in \mathcal{R}$, then r_n is *indirectly accessible* for r_1.

B. The Dependency Graph

A *dependency graph* of a system with the set of resources $\mathcal{R}$ is a directed graph $\hat{G} = (\mathcal{R}, \hat{E})$ with $\hat{E} \subseteq (\mathcal{S} \times \mathcal{S} \cup \mathcal{U} \times \mathcal{S})$. $\hat{G}$ contains an edge (r,s) whenever a resource r depends on the resource s concerning its availability. In other terms, r needs accessibility to s. The edges in $\hat{E}$ are again labeled with the subjective weight $w(r,s)$ of resource s for r.

Fig. 1 depicts the dependency graph of an arbitrary Voice-over-IP (VoIP) application and its different sub-compontents (e.g. recording/replaying, buffering, encoding/decoding, sending/receiving, session management) from access points and subsystems of the operating system, its hardware drivers and finally the hardware itself.

When modelling multiple systems in a network, we observe different classes of dependencies between the resources. The first class covers *mandatory* dependencies, where the availability of a resource r is immediately affected when either the availability or direct accessibility of the resource r which it directly depends on is limited. All dependencies from the example in Fig. 1 are mandatory, e.g. if the audio hardware is only available up to a limited extent, the record/replay components are affected. The set of resources on which a resource r depends on mandatorily is denoted as $\mathcal{R}^{(1)}(r) \subset \mathcal{R}$.

Other classes of dependencies to which r may belong include:

- *Alternative* (denoted as $\mathcal{R}^{(2)}(r) \subset \mathcal{R}$): The availability of a resource r is immediately affected when the availability or direct accessibility of all resources it directly depends on is limited. If at least one of these resources is fully available, r is fully available as well. An example is the dependency of a web content provider from multiple server hosts on a webserver cluster, where one server is enough to provide the content.
- *Combined* (denoted as $\mathcal{R}^{(3)}(r) \subset \mathcal{R}$): The availability of a resource r is immediately affected when the availability or direct accessibility of all resources it directly depends on is limited. r is as available as the average of these resources. An example is a user who simultaneously depends on a navigation system and on a voice communication application.
- *m-out-of-n* (denoted as $\mathcal{R}^{(4)}(r) \subset \mathcal{R}$): The availability of a resource r is fully given if at least m from n resources r depends on, are fully available and directly accessible. An example is a distributed filesystem, where a given number of storage entities needs to be accessible.
- *Indirect* (denoted as $\mathcal{R}^{(5)}(r) \subset \mathcal{R}$): The availability of a resource r is immediately affected when the availability or direct accessibility of at least one resource it directly or indirectly depends on, is limited. An example is the dependency of an instance of an IP transport service (including routing and forwarding) from the other instances in the network. The more instances are directly or indirectly (with other instances as mediators) accessible, the better is the coverage of the network.

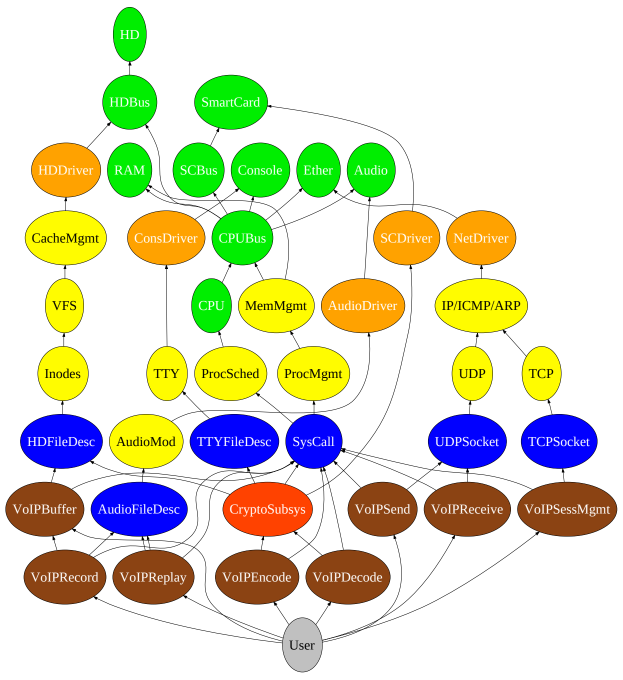

Fig. 1. Dependency Graph of a VoIP application, the operating system and hardware components. Edges indicate mandatory dependencies between the connected nodes with a weight of one.

Fig. 2 depicts the availability dependencies of a mobile adhoc network (MANET) with three identical nodes which consist of instances of basic hardware/operating services (OS), MANET networking/routing services (Network) and two applications, namely a multicast Voice-over-IP (VoIP) and a fully meshed Command and Control System (C2S). Mandatory dependency edges are depicted as solid arrows, optional dependencies as dotted arrows, and indirect dependencies as dashed arrows. All edges have a weight of one; edge labels are omitted.

Note that both network and C2S service instances are comprising multiple nodes on a single host, since their dependencies belong to different dependency classes (e.g. the local navigation and the communication module for the C2S nodes). This is not the case for the OS and VoIP nodes.

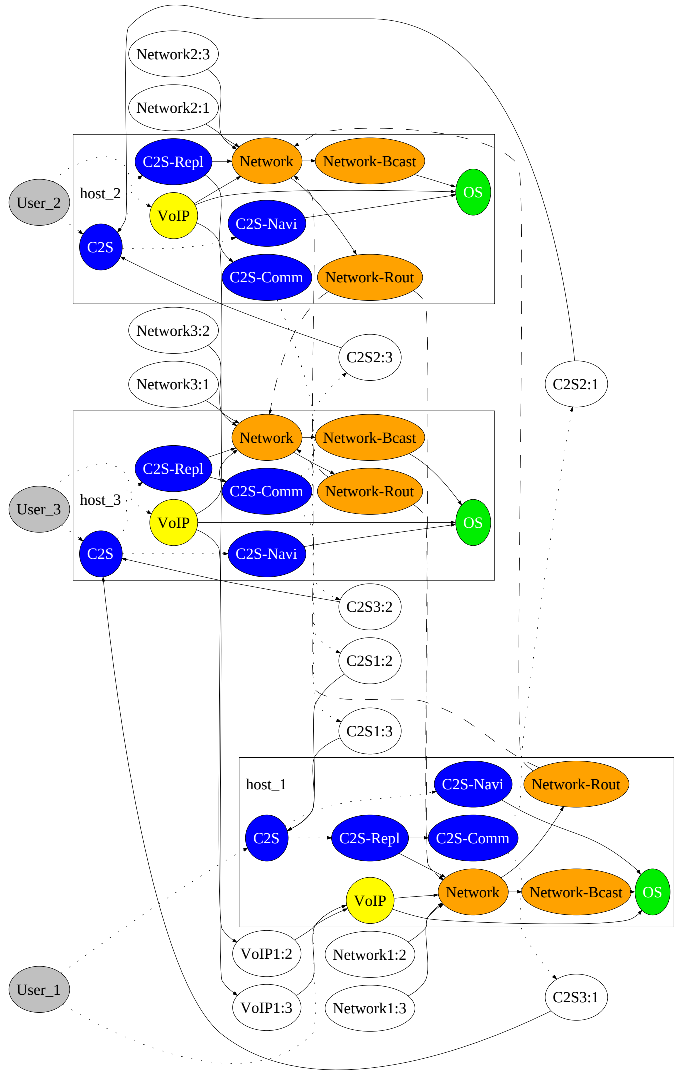

Fig. 2. Dependency graph of a simple MANET with three hosts (see text).

In this example, a user is assigned to one single host in this network and he is depending on both applications to an equal extent. The networking/routing service instances indirectly depend on all other instances. The C2S communication instances depend on the other instances in combination. Concerning the VoIP application, host 1 acts as a multicast VoIP sender, so the instances on hosts 2 and 3 depend on the VoIP instance of host 1.

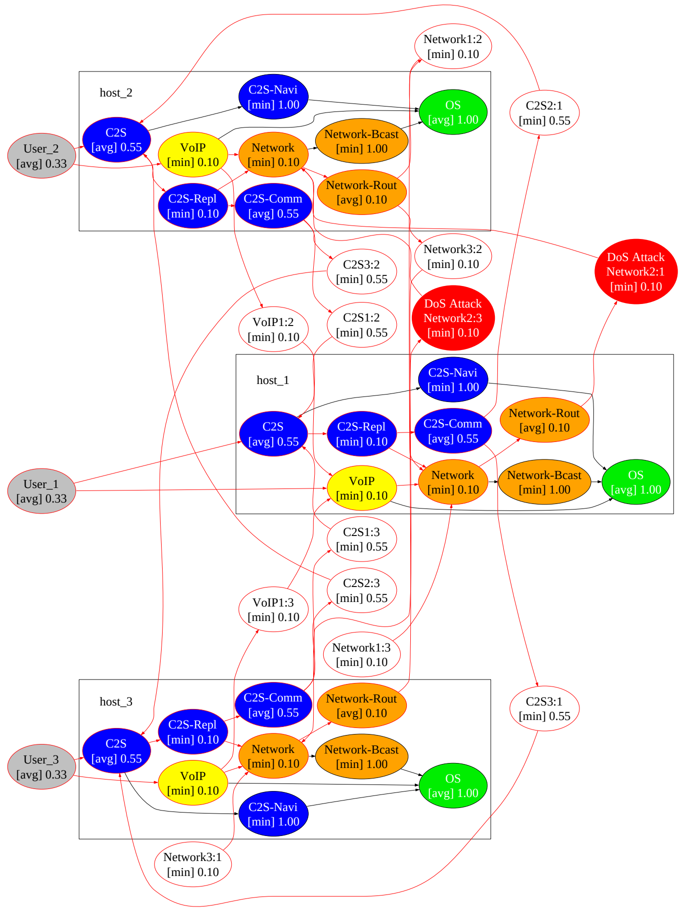

Fig. 3. Accessibility Graph after DoS attack against networking/routing service instance #2. Nodes Network2:1 and Network2:3 have a decreased availability of 0.10 each; all dependent nodes have been updated.

C. The Accessibility Graph

An *accessibility graph* of a system with the set of resources $\mathcal{R}$ is a directed graph $G = (\mathcal{R}, E)$ with $E \subseteq (\mathcal{S} \times \mathcal{S} \cup \mathcal{U} \times \mathcal{S})$. G contains an edge (r, s) whenever a resource r has direct access to s. Here 'direct' means that no other resources act as mediators for the access. Thus, one will not see edges in our accessibility graph G, that are representing indirect dependencies in the dependency graph $\hat{G}$.

The nodes $r \in \mathcal{R}$ of the accessibility graph G are labeled with their availability $A(r)$, and the edges $(r, s) \in E$ are labeled with the subjective weight $w(r, s)$ of node s for r (i.e. its relative importantness compared to the other nodes r depends on).

Fig. 3 shows an example for an attack graph, corresponding to the above example dependency graph. The network/routing service instances are connected in a daisy chain. A DoS attack was launched against the network service on node #2, so that only 10% of the packets coming from nodes #1 and #3 are processed, as indicated by a diagnosis system, resulting in a lowered overall availability value of 0.33.

D. Determining the Resource and Overall Availability

Each time the diagnosis systems indicate a changed availability of a resource, the availability of resources which directly or indirectly depend on the changed one, need to update their values immediately. As mentioned, the propagated availability values $A_P(r)$ need to be updated, the intrinsic values $A_I(r)$ stay as they are as long as there is no indication for a change.

To incorporate all potential dependencies of a node, we define the propagated availability, depending on the class of dependencies, as follows:

1) *Mandatory* dependencies: To model the fact that the availability of a resource r cannot exceed the value of the resources it depends on, the minimum operator has been chosen:
$$A_P(r) := \min_{s \in \mathcal{R}^{(1)}(r)} A(r,s)$$
2) *Alternative* dependencies: To reflect that at least the availability of one of the resources that r depends on, is sufficient to be available, the maximum operator has been chosen:
$$A_P(r) := \max_{s \in \mathcal{R}^{(2)}(r)} A(r,s)$$
3) *Combined* dependencies: To model the dependency of the availability from the average of the other resources, the arithmetic mean has been chosen:
$$A_P(r) := \frac{\sum_{s \in \mathcal{R}^{(3)}(r)} A(r,s)}{|\mathcal{R}^{(3)}(r)|}$$
4) m*-out-of-*n dependencies: To take into account that r is only fully available if at least m from n resources r depends on, are fully available and directly accessible, the operator needs to be as follows:
$$A_P(r) := \begin{cases} 1 & \text{if } \exists a(r) \subseteq \mathcal{R}^{(4)}(r), \text{with } |a(r)| \geq m \\ & \text{s.t. } A(s) = 1 \ \forall s \in a(r) \\ 0 & \text{else} \end{cases}$$
Note that the parameter m needs to be specified, and the weights $w(r,s)$ need to be identical equal to one.
5) *Indirect* dependencies: To incorporate all influences of other resources r depends on equally, the average operator has been chosen. Note that in this case, the intrinsic availability of r has not been considered by intention.
$$A_P(r) := \frac{\sum_{s \in \mathcal{R}^{(5)}(r)} A'_P(r,s)}{|\mathcal{R}^{(5)}(r)|}$$

For direct accessibility – which needs to be respected for classes #1-#4 – the propagated value of a single graph edge shall be given as
$$A(r,s) := w(r,s) \cdot A(s) \tag{2}$$
where $A(s)$ is the availability of the node s that r depends on, and $w(r,s) \in [0,1]$ is the subjective weight of s for r.

For access with other resources as mediators (class #5), we need to determine the availability, depending on whether there exists a path from $r_0 := r$ to $r_n := s$ in G:
$$A'_P(r,s) := \max_{\substack{((r_0,r_1),\ldots,(r_{n-1},r_n)) \\ (r_{i-1},r_i) \in E \ \wedge \ A(r_i) > 0 \\ \forall 0 < i \leq n}} w(r,s) \cdot A(r_1) \tag{3}$$

Finally, to quantify the overall availability of the network in the light of supporting users when conducting a mission, a corresponding definition is needed. Intuitively, defining the overall availability as the availability of the service instances that are immediately needed by the users is useful. So we define the overall availability as
$$A(G) := \frac{\sum_{u \in \mathcal{U}} A(u) \cdot m(u)}{\sum_{u \in \mathcal{U}} m(u)} \tag{4}$$
where $m(u) \in [0,1]$ is the *relative importance* of the user $u \in \mathcal{U}$ for the common mission, that needs to be defined beforehand[1].

Given that all users gain equal importance (i.e. 1.0) for the example graph Fig. 3, the resulting overall availability is
$$A(G) = \frac{3 \cdot 0.33}{3} = 0.33$$
This value reflects that this is a serious attack that results in a limited value of the network for all application instances to support the mission.

E. Updating the Accessibility Graph

The idea is to use the availability values that are propagated to the users as overall quantifiers for the availability of the network. This definition is intuitive, since all the goals are aligned to the mission of the users that is supported by the network.

For each availability changed, the values need to be propagated to all affected resources in the network. For achieving this, an algorithm with the following properties is needed:

- It needs to capture all affected nodes.
- It terminates, even if there are cyclic dependencies in the accessibility graph which cannot be precluded.
- It needs to give stable results, i.e. multiple subsequent applications should yield identical values.

A possibility to fulfil most of the requirements is based on an inverse breadth-first-search (BFS) in a directed graph:

```
1: for all nodes $r$ with a changed intrinsic availability $A_I(r)$
   or with a new or removed edge $(r,s)$ do
2:     $enqueue(Q, r)$
3: end for
4: for all nodes $r$ from $\mathcal{R}$ do
5:     determine their shortest distance $d[r]$ to nodes from $Q$
       by performing a BFS, starting from nodes in $Q$
6:     $color[r] \leftarrow$ WHITE
7: end for
```

[1]Currently, inter-dependencies between weights are not considered.

```
8:  while $Q \neq \emptyset$ do
9:    $r \leftarrow dequeue[Q]$
10:   for all nodes $s$ from $parent[r]$ do
11:     if $d[s] \geq d[r] \vee (class[r] == \text{USER}$
        $\wedge\ color[s] == \text{WHITE} \wedge d[s] \geq 0)$ then
12:       $d[r] := d[r] + 1$
13:       $enqueue(Q, r)$
14:       goto step 9
15:     end if
16:   end for
17:   if $color[r] \neq \text{BLACK}$ then
18:     Calculate $A_P(r)$ as defined above
19:     $color[r] \leftarrow \text{BLACK}$
20:   end if
21:   if $d_l < d[r]$ then
22:     $A(r) = A_I(r) \cdot A_P(r)$
23:   end if
24:   for all nodes $s$ from $children[r]$ do
25:     if $color[s] == \text{WHITE}$ then
26:       $enqueue(Q, s)$
27:       $color[s] \leftarrow \text{GREY}$
28:     end if
29:   end for
30:   $d_l \leftarrow d[r]$
31: end while
```

This algorithm traverses the accessibility graph, starting at the nodes with changed availabilities and updates the values of their children. This behaviour guarantees the completeness and the termination. The coloring of the nodes indicates their processing status: BLACK nodes are updated, GREY nodes are in the process queue, and WHITE nodes are not enqueued. Cyclic dependencies are ignored during the traversal process. Therefore, the algorithm works comparable to the software reliability risk analysis algorithm from [YCA04].

A significant improvement is the roundwise update of calculated availability values (step 22) as well as postponing the update of nodes with parents who have a bigger or equal distance to the start nodes (steps 10-16). This is needed to process parallel dependency paths in a stable manner.

F. Definition and Example for Reaction Success

To be able to assess properties of a reaction, an accessibility graph for the system state prior and after the application needs to be generated (by knowledge about the earlier state and influenced by the current output of the diagnosis systems).

An example of this pair of graphs is the one from Figs. 3 and 4. The earlier one describes the situation after an attack and the latter one shows the effects of the applied reaction. In this case, the reaction is a change of the routing tables on hosts #1 and #3, so that host #2 that was affected by the attack in Fig. 3, was separated from the network.

Finally, if we assume that G is the graph we obtain before the reaction, and G' after the reaction, the *success* of the reaction can be intuitively defined as the change of availability after the reaction against an attack:

$$\delta_1(G, G') := A(G') - A(G) \tag{5}$$

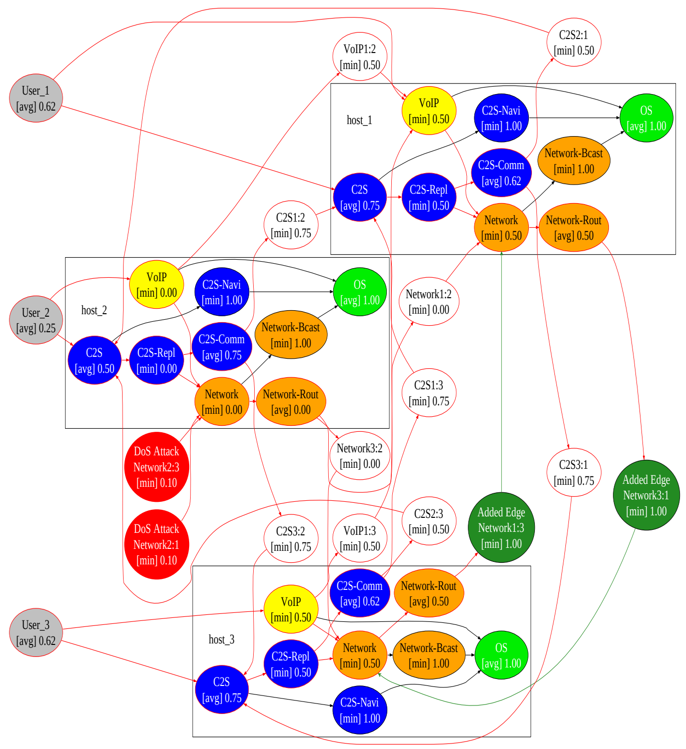

Fig. 4. Accessibility Graph after separating the attacked host #2 using routing reconfiguration as a response measure. Edges (Network-Rout_1,Network3:1) and (Network-Rout_3,Network1:3) have been added instead of (Network-Rout_1,Network2:1) and (Network-Rout_3,Network2:3), respectively.

Obviously, this metric may also have negative values, since a wrong selection of response measures might also damage the network rather than having a positive effect. This matches also with the intuition.

For the example above, the according availabilities have been $\frac{0.62+0.25+0.62}{3}$ and 0.33, respectively. Thus, the success of the reaction with its results depicted in Fig. 4 can be quantified as

$$\delta_1(G, G') \approx 0.50 - 0.33 = 0.17$$

G. Additional Reaction Metrics

Besides the success of a reaction, other (secondary) metrics are important for estimating the value of a reaction and for selecting responses for application, as discussed in section III. So, our second metric is the effort that is needed for implementing a reaction.

Thanks to the simple model we use, it is possible to define the *effort* of a reaction as the normalized number of resources (service instances) which need to be modified in order to implement the reaction:

$$\delta_2(G, G') := \frac{\#\text{of modified instances}}{|\mathcal{S}|} \tag{6}$$

Note that the orientation of the effort metric is different from the success metric, since a bigger success is 'better' than a smaller value, whereas a smaller effort is 'better' than a bigger value.

For the example discussed in the last subsection, the effort was given as the number of network/routing instances that needed to modify their routing tables. Thus the resulting effort value was

$$\delta_2(G, G') = \frac{4}{38} \approx 0.11$$

To see the influence of the combination of both defined metrics, we introduce the example of another possible reaction for the same attack. As depicted in Fig. 5, it seems to be also possible to preclude the attacked host #2 from access to the remote application services rather than separating it from the network.

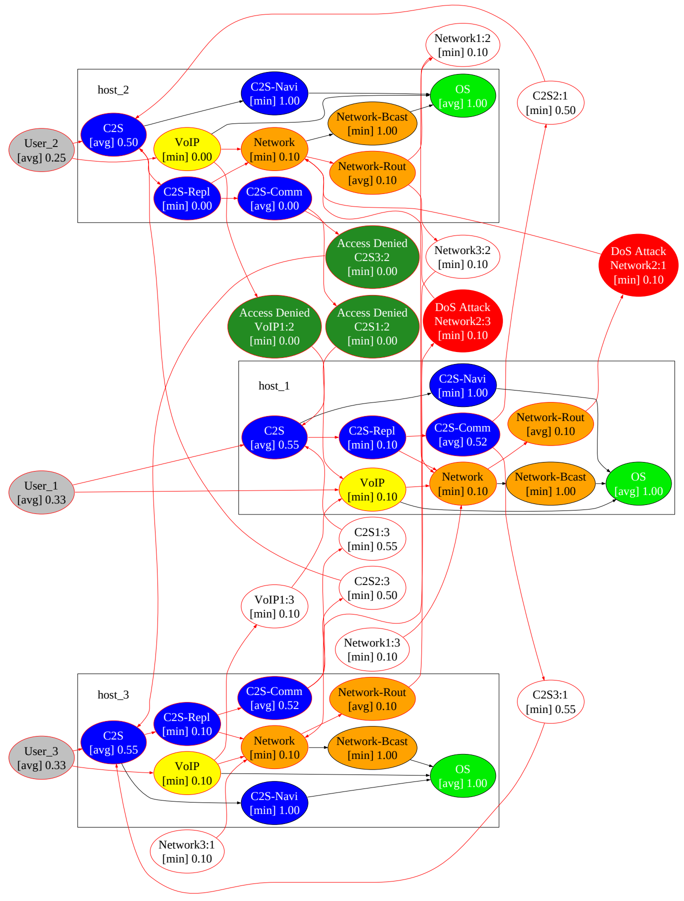

Fig. 5. Accessibility Graph after separating the attacked host #2 using service exclusion as an alternative response measure. Availability of nodes C2S1:2, C2S3:2, and VoIP1:2 has been set to zero.

The obtained results for the success and for the effort of this alternative reaction (resulting in another availability graph G'') are

$$\delta_1(G, G'') \approx \frac{0.33 + 0.25 + 0.33}{3} - 0.33 = -0.3$$

and

$$\delta_2(G, G'') = \frac{3}{38} \approx 0.08$$

This alternative reaction yields a negative result for the success metric. The reason is that precluding the attacker from applications has no impact for the network connectivity. Thus, concerning availability, the earlier reaction would be considered the more appropriate solution, even if the estimated effort is lower than for the alternative.

Currently, definitions for two other important metrics (*error-proneness* and *durability*) using properties of the graphs are under examination.

V. Discussion

In this stage of developing our approach, many aspects are under discussion, as the following subsections evince.

A. Advantages and Disadvantages

The graph based approach as described in this contribution has a number of noteworthy advantages, including a large degree of intuitivity, the ability to be adapted for differently scaled and granulated scenarios as well as relatively easy maintainability. These are basically results of the fact that the graph model acts as a 'small world' picture with graph properties that cover important aspects of the real world system which is to be protected.

On the other hand, the way to obtain a 'state zero' graph and to modify it by creating a mapping from the output of the diagnosis systems (NMS, IDS/IPS) to the availability values incorporates potential difficulties, since this mapping must be created beforehand and needs to have a sufficient degree of reliability and consistency. However, it seems feasible to obtain this mapping, as recent experiments have indicated.

Another negative property of our approach is the fact, that cyclic structures in the dependency and accessibility graphs cannot be precluded. This is due to the existence of cyclic dependencies in real world systems. The way we worked around this problem by breaking up these cycles currently has the disadvantage that it does match intuitivity only to a certain extent. We are currently investigating mechanisms for dealing with cyclic dependencies.

B. Comparison to Plan Recognition Oriented Approaches

Many researchers (e.g. Geib and Goldman [GG01], Yu and Frincke [YF07]) are focusing on a different approach for selecting response actions, namely the 'plan recognition' or 'goal prediction' type approach. Most of this work focuses on identifying a hostile agents plan and predicting his further actions in order to select an appropriate defending strategy. Approaches of this type aim at obtaining a more holistic view on attackers.

But modelling attacker plans and goals in our type of situational graphs would lead to a large complexity when obtaining, storing and updating the data structures. Additionally, the high degree of intuitivity of the model disappears, and thus the ease of maintainability would also be affected. It also would bring probabilities back to the table which we tried to avoid on purpose, since probabilistic evaluations would suppose the NSO to have an expert a priori knowledge of e.g. probabilities of attack steps. Nevertheless, it is clear that most attacks are related and/or are part of a plan to achieve a top level goal.

All in all, we believe that putting an appropriately selected response for observed symptoms in place is still the most effective measure, since dangerous attackers would probably not follow predictable plans or standard procedures.

However, plan recognition approaches might be used to optimize our graph model, e.g. for determining values for error-proneness and durability metrics.

C. Extending the Approach: From Availability to Confidentiality

As the whole description of the approach was focused on availability, a way to extend it to other IT security objectives is needed. We propose to add more layers to the dependency graph which each comprise a subgraph that represents e.g., the degree of integrity. To reflect this, each layer subgraph needs to have an identical set of nodes which are associated with the current integrity values of the respective resources.

Intuitively, most of the security properties rely on others. Thus, additional inter-layer dependencies may be introduced in the multi-layered graph, such as depicted in Fig. 6. Intuitively, all higher layer nodes maintain dependencies to the respective nodes on the lowest (availability) layer, since properties like integrity and confidentiality of a resource may only be verified, if the resource itself is available.

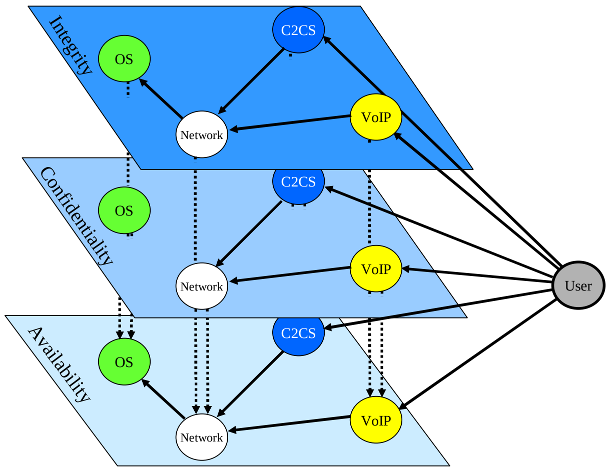

Fig. 6. Multi-layered dependency graph for expressing multiple IT security relevant resource properties.

VI. Conclusion and Further Work

This contribution proposes a graph based approach for estimating properties of reactions for given attacks against computer systems and networks. Thus, the approach can be used for both quantifying the effects of a response measure after its application and for choosing the most promising alternative of a set of available response measures. For the time being, only availability as one of the most important IT security properties is in the focus of the modelling approach. For the observed system, a dependency graph reflects the inter-dependencies of the system resources (service instances and users). For the current state of the system, an accessibility graph depicts the momentary availabilities of the resources and their accessibilities for each other. The accessibility graph is a result of the output of different diagnosis systems, such as intrusion detection or network management systems. An update algorithm calculates the effects on all directly or indirectly dependent resources and also on the users.

By comparing properties of accessibility graphs prior and after a reaction has been applied to the system, pragmatically important aspects like response success, application effort, error-proneness and durability can be quantified. Different examples have underlined the applicability of this approach in different scenarios with different granulatities.

This contribution also discussed advantages and disadvantages of the approach, especially in the light of other relevant work. Several aspects to be investigated further have been pointed out. These future work areas include graph properties for the error-proneness and durability metrics and plausibility investigations for the arithmetic operations that reflect the inter-resource dependencies. The update algorithm will be analyzed, optimized and extended for handling cyclic dependencies, up to a certain extend. Additionally, we plan to implement and evaluate the model in a MANET environment with precisely defined properties.

References

[SBW07] N. Stakhanova, S. Basu, and J. Wong. A Taxonomy of Intrusion Response Systems. International Journal of Information and Computer Security Vol. 1(1), pp. 169–184, 2007.

[Den99] D. Denning. Information Warfare and Security. Addison-Wesley, 1999.

[Nor99] S. Northcutt. Intrusion Detection: An Analyst's Handbook. New Riders Publishing, 1999.

[LFM+02] W. Lee, W. Fan, M. Miller, and S. Stolfo. Toward Cost-Sensitive Modeling for Intrusion Detection and Response. Journal of Computer Security Vol. 10, pp. 5–22, 2002.

[TC02] T. Toth and C. Kruegel. Evaluating the impact of automated intrusion response mechanisms. In: Proc. of the 18th Computer Security Applications Conference (ACSAC'02), pp. 301–310, Las Vegas, NV, USA, 2002.

[BMR+03] I. Balepin, S. Maltsev, J. Rowe, and K. Levitt. Using specification-based intrusion detection for automated response. In: Proc. of the 6th International Symposium on Recent Advances in Intrusion Detection (RAID 2003), 2003.

[WFB+07] Y. Wu, B. Foo, Y. Mao, S. Bagchi, and E. Spafford. Automated adaptive intrusion containment in systems of interacting services. Computer Networks Vol. 51(5), pp. 1334–1360, 2007.

[YCA04] S. Yacoub, B. Cucik, and H. Ammar. A Scenario-Based Reliability Analysis Approach for Component-Based Software. IEEE Transactions on Reliability, Vol. 53(4), 2004.

[YA02] S. Yacoub and H. Ammar. A Methodology for Architectural-Level Reliability Risk Analysis. IEEE Transactions on Software Engineering, Vol. 28(6), 2002.

[Sno07] The Snort Inline Project Team. Snort Inline Project Homepage. Online accessible at http://snort-inline.sourceforge.net/, 2007.

[GG01] C. Geib and R. Goldman. Plan Recognition in Intrusion Detection Systems In: Proc. of the DARPA Information Survivability Conference and Exposition - II (DISCEX-II), 2001.

[YF07] D. Yu and D. Frincke. Improving the quality of alerts and predicting intruder's next goal with Hidden Colored Petri-Net. Computer Networks Vol. 51(3), pp. 632–654, 2007.