Abstraction-based analysis of known and unknown vulnerabilities of critical information infrastructures

Roland Rieke

Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany
rieke@sit.fraunhofer.de

Abstract. The systematic protection of critical information infrastructures requires an analytical process to identify the critical components and their interplay, to determine the threats and vulnerabilities, to assess the risks and to prioritise countermeasures where risk is unacceptable. The abstraction-based approach presented here builds on a model-based construction of an attack graph with constraints given by the network security policy. A unique feature of the presented approach is, that abstract representations of these graphs can be computed that allow comparison of focussed views on the behaviour of the system. In order to analyse resilience of critical information infrastructures against exploits of unknown vulnerabilities, generic vulnerabilities for each installed product and affected service are added to the model. The reachability analysis now considers every possible choice of product, and so all alternatives are evaluated in the attack graph. The impact of changes to security policies or network structure can be visualised by differences in the attack graphs. Results of this analysis support the process of dependable configuration of critical information infrastructures.

Key words: threats analysis, attack simulation, critical infrastructure protection, network security policies, risk assessment, security modelling and simulation, unknown vulnerabilities.

1 Introduction

Information and Communication Technology (ICT) is creating innovative systems and extending existing infrastructure to such an interconnected complexity that predicting the effects of small internal changes (e.g. firewall policies) and external changes (e.g. the discovery of new vulnerabilities and exploit mechanisms) becomes a major problem. The security of such a complex networked system essentially depends on a concise specification of security goals, their correct and consistent transformation into security policies and an appropriate deployment and enforcement of these policies. This has to be accompanied by a concept to adapt the security policies to changing context and environment, usage patterns and attack situations. To help to understand the complex interrelations of security policies, ICT infrastructure and vulnerabilities and to validate security goals in such a setting, tool-based modelling techniques are required that can efficiently and precisely predict and analyse the behaviour of such complex interrelated systems. Figure 1 shows an example of such an infrastructure. Known and unknown vulnerabilities may be part of each of the connected components and communication paths between them. Analysis methods should guide a sys-

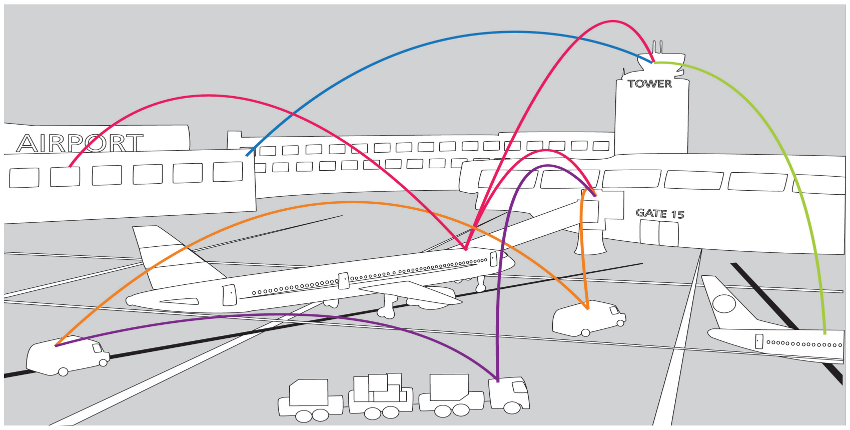


Fig. 1. Interplay of complex interrelated systems

tematic evaluation of such a critical information infrastructure assist the persons in charge with finally determining exactly how to configure protection measures and which security policy to apply.

A typical means by which an attacker (directly or using malware such as blended threats) tries to break into such a network is, to use combinations of basic exploits to get more information or more credentials and to capture more assets step by step. To find out if there is a combination that enables an attacker to reach critical network resources or block essential services, it is required to analyse all possible sequences of basic exploits, so called *attack paths*.

For this type of analytical analysis, a formal modelling framework is presented that, on the one hand, represents the information system and the security policy, and, on the other hand, a model of attacker capabilities and profile. It is extensible to comprise intrusion detection components and optionally a model of the system's countermeasures. Based on such an operational model, a graph representing all possible attack paths can be automatically computed. It is called an *attack graph* in the following text. Based on this attack graph, it is now possible to find out whether a given security policy successfully blocks attack paths and is robust against changes in the given vulnerability setting.

One problem now is, that it is usually impossible to visualise an attack graph of a realistic example directly because of the huge size. However, abstract representations of an attack graph can be computed and used to visualise and analyse compacted information focussed on interesting aspects of the behaviour. The impact of changes to security policies can be visualised by finding differences in the attack graphs or the abstract representations thereof.

This paper also shows, that abstract representations are very useful to analyse resilience of critical information infrastructure with respect to attacks based on unknown vulnerabilities because addition of unknown vulnerabilities results in very large attack graphs.

The subsequent paper is structured as follows. The modelling approach is described in Sect. 2, while Sect. 3 presents an exemplary analysis. Section 4 presents an approach to analyse resilience of critical information infrastructures against exploits of unknown vulnerabilities. Section 5 gives an overview of related work. Finally, this paper ends with an outlook in Sect. 6.

2 Modelling information infrastructures and threats

The proposed operational model comprises, (1) an asset inventory including critical network components, topology and vulnerability attributions, (2) a network security policy, (3) vulnerability specifications and exploit descriptions, and (4) an attacker model taking into account the attackers knowledge and behaviour.

2.1 ICT network components

The set of all hosts of the information system consists of the union of the hosts of the ICT network and the hosts of the attacker(s). Following the M2D2 model [1], *products* are the primary entities that are vulnerable. A *host configuration* is a subset of products that is installed on that host and *affects* is a relation between vulnerabilities and sets of products that are affected by a vulnerability. A host is *vulnerable* if its configuration is a superset of a vulnerable set of products and the affected services are currently running. In order to conduct a subsequent comparative analysis of attack paths, an asset prioritisation as to criticality or worth regarding relative importance of the assets is required.

2.2 Network security policies

The model of the network security policies is based on the Organisation-Based Access Control (Or-BAC) model [2]. The advantage of this choice is, that it is possible to link the policies in the formal model at an abstract level to the low level vendor specific policy rules for the Policy Enforcement Points (PEPs) such as firewalls in the concrete ICT network.

To illustrate the modelling concepts described here, a small example scenario is given in Fig. 2. Modelling concepts and typical analysis outcome will be illustrated using this example scenario throughout the paper. One possible attack path is sketched in the scenario.

Following the Or-BAC-based concept, the network vulnerability policy is given at an abstract level in terms of *roles* (an abstraction of subjects), *activities* (an abstraction of actions) and *views* (an abstraction of objects). A *subject* in this model is any host. An *action* is a network service such as snmp, ssh or ftp. Actions are represented by a triple of protocol, source port and target port. An *object* is a message sent to a target host. Currently only the target host or rather the role of the target host is used for the view definition here. To specify the access control policy using this approach, *permissions* are given between role, activity and view. For the example scenario the hostnames *telework*, *attacker*, *nix_host*, *ms_host*, *db_server* and *portal* are used. The roles of these hosts are given by the table in Fig. 3(a). The policy permissions are defined by the table in Fig. 3(b).

Mobile components. To model mobile components that can transport malware such as blended threats from one network zone to another, it is convenient to allow a host to play

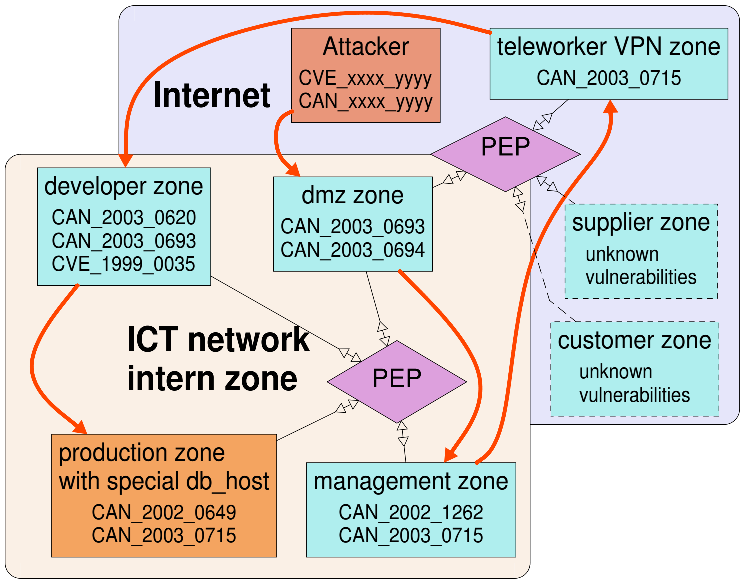


Fig. 2. ICT network and vulnerabilities

different roles. For example in Fig. 3(a) the host *telework* that plays the role *telework_host* can additionally be permitted to play the role *intern_host*. In this case an attack path could cross the zones from *telework_host* to *intern_host* without any restrictions by the network security policy. Similar problems exist in infrastructures with mobile components such as the example scenario shown in Fig.1.

2.3 Vulnerabilities

Vulnerability specifications for the formal model of the example scenario are derived from the Common Vulnerabilities and Exposures (CVE/CAN) descriptions. The MITRE Corporation provides a list of virtually all known vulnerabilities (`http://www.cve.mitre.org/`). The CVE name is the 13 character ID used by the CVE standards group to uniquely identify a vulnerability. Additional information about the vulnerabilities also covers preconditions about the target host as well as network preconditions. Furthermore, the impact of an exploitation of a vulnerability is described. The specifications for the formal model of the vulnerabilities additionally comprise the vulnerability *range* and *impact type* assessments provided by the National Institute of Standards and Technology (NIST) (`http://nvd.nist.gov/`). Of course, other kinds of vulnerabilities could be added to the model in a similar manner.

Vulnerability severity. The Common Vulnerability Scoring System (CVSS) [3] provides universal severity ratings for security vulnerabilities. These ratings are used in the model as an example for a measure of the threat level. Another example for such a measure is the metric used by the US-CERT (cf. `http://www.kb.cert.org/vuls/html/fieldhelp#metric`).

Role	Hosts
internet_host	*attacker*
dmz_host	*portal*
telework_host	*telework*
developer_host	*nix_host*
management_host	*ms_host*
db_host	*db_server*
intern_host	*db_server*, *ms_host*, *nix_host*

(a) Roles

Role (source)	View (target)	Activity (service)
internet_host	*internet_host*	*any_activity*
any_role	*dmz_host*	*ssh*
any_role	*dmz_host*	*smtp*
dmz_host	*intern_host*	*ssh*
intern_host	*any_role*	*net*
intern_host	*internet_host*	*ftp*
intern_host	*internet_host*	*rsh*
intern_host	*dmz_host*	*ssh*
db_host	*production_host*	*rpc*
teleworker_host	*dmz_host*	*any_activity*

(b) Network security policy

Fig. 3. Roles and network security policy

These measures are based on information about the vulnerability being widely known, reported exploitation incidents, number of infected systems, the impact of exploiting the vulnerability and the knowledge and the preconditions required to exploit the vulnerability. Because the approximate values included in those measures may differ significantly from one site to another, prioritising of vulnerabilities based on such measures should be used with caution.

To have a vulnerable product installed on some host, does not necessarily imply, that someone can exploit that vulnerability. A target host *is configured vulnerable*, if (1) the target host has installed a product or products that are vulnerable with respect to the given vulnerability, and (2) necessary other preconditions are fulfilled (e.g. some vulnerabilities require that a trust relation is established as for example used in remote shell hosts allow/deny concepts).

The second precondition to exploit a vulnerability is, that the target host *is currently running the respective products* such as a vulnerable operating system or server version. If a user interaction is required this also requires that the vulnerable product is currently used (e.g. a vulnerable Internet explorer).

The third necessary precondition is, that the *network security policy permits* that the target host is reachable on the port the vulnerable product is using from the host the attacker selected as source.

2.4 Attacker profile

The knowledge of exploits and hosts and the credentials on the known hosts constitute an attackers profile. Knowledge about hosts changes during the computation of the attack graph because the attacker might gain new knowledge when capturing hosts. On the other hand, some knowledge may become outdated because the enterprise system changes ip-numbers or other configuration of hosts and reachability. In case a vulnerability is exploited, the model has to cover the *effects for the attacker* (e.g. to obtain additional user or root credentials on the target host) and also the *direct impact on the network and host* such as, to shut down a service caused by buffer overflow.

2.5 Formal representation of the model

The information model presented so far covers the description of a (static) configuration of an ICT network and its vulnerabilities. In the formal model such a configuration describing the *state* of the ICT network is represented by *APA state components*.

Definition 1. *An* Asynchronous Product Automaton (APA) *consists of*

- *a family of* state sets $(Z_s)_{s\in\mathbb{S}}$,
- *a family of* elementary automata $(\Phi_e, \Delta_e)_{e\in\mathbb{E}}$ *and*
- *a* neighbourhood relation $N : \mathbb{E} \to \mathcal{P}(\mathbb{S})$
- *an* initial state q_0

$\mathbb{S}$ *and* $\mathbb{E}$ *are index sets with the names of state components and of elementary automata and* $\mathcal{P}(\mathbb{S})$ *is the power set of* $\mathbb{S}$.

For each elementary automaton (Φ_e, Δ_e) *with* Alphabet Φ_e*, its* state transition relation *is* $\Delta_e \subseteq \bigtimes_{s\in N(e)}(Z_s) \times \Phi_e \times \bigtimes_{s\in N(e)}(Z_s)$*. For each element of* Φ_e *the state transition relation* Δ_e *defines state transitions that change only the state components in* $N(e)$*. An APA's (global)* states *are elements of* $\bigtimes_{s\in\mathbb{S}}(Z_s)$*. To avoid pathological cases it is generally assumed that* $\mathbb{S} = \bigcup_{e\in\mathbb{E}}(N(e))$ *and* $N(e) \neq \emptyset$ *for all* $e \in \mathbb{E}$*. Each APA has one* initial state $q_0 = (q_{0s})_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$*. In total, an APA* $\mathbb{A}$ *is defined by*

$$\mathbb{A} = ((Z_s)_{s\in\mathbb{S}}, (\Phi_e, \Delta_e)_{e\in\mathbb{E}}, N, q_0)$$

Finite state model of the scenario. The components of the model described previously are now specified for the proposed analysis method using the *APA state components* $\mathbb{S} = \{$ A_known_exploits_state, A_plvl_state, affects_state, configuration_state, host_service_state, host_vulnerability_state, host_vulnerable_user_state, ... $\}$.

The initial state is composed of $q_{0A\_known\_exploits\_state}$, $q_{0A\_plvl\_state}$, ..., where $q_{0A\_known\_exploits\_state}$ contains the exploits known by the attacker,$q_{0A\_plvl\_state}$ contains a sequence of pairs of host and access privileges of the attacker on that host (e.g. (attacker,root), (db_server,none), ...),$q_{0affects\_state}$ contains a sequence of pairs of vulnerability and affected product (e.g. (CAN_2002_0649,SQL_Server_2000), (CAN_2002_1262,vulnerable_IE), ...), $q_{0configuration\_state}$ contains a sequence of pairs of a host and a sequence of installed products (e.g. (db_server,W2000_Server.SQL_Server_2000), ...),$q_{0host\_service\_state}$ contains a sequence of pairs of host and associated service including used port and privileges (e.g. (db_server,((ftpd,ftp_port),root)), (db_server, ((sql_res, ms_sql_m_port), db_user)). ...), and, $q_{0host\_vulnerability\_state}$ which is empty (the vulnerabilities are computed from $affects\_state$ and $configuration\_state$ in a preprocessing transition).

To describe how actions of attacker(s) and actions of the system can change the state of the ICT network model, specifications of *APA state transitions* are used. These state transitions represent atomic exploits and optionally the actions that the system executes itself (e.g. to implement vital services).

The set of *elementary automata* $\mathbb{E} = \{$ Preprocessor_gen_vulnerabilities, Service_answer, A_select_exploit, Defence_Restart_sshd, A_CAN_2002_1262_IE_caching_exploit, A_CAN_2002_0649_sql_exploit, A_CAN_2003_0694_sendmail_exploit, ... $\}$ represents the possible actions. The actions starting with $A\_\ldots$ are the actions the attacker can perform. If multiple attackers are modelled then A is replaced by the name of the attacker.

A state transition can occur, when all expressions are evaluable and all conditions are satisfied. So called *interpretation variables* are used to differentiate the variants of execution of the same transition. All possible variants of bindings of interpretation variables from the state components are generated automatically. So for example for a transition modelling an exploit, all possible combinations of bindings of source and target host are computed and further evaluated.

Definition 2. *An elementary automaton* (Φ_e, Δ_e) *is* activated *in a state* $q = (q_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ *as to an* interpretation $i \in \Phi_e$*, if there are* $(p_s)_{s\in N(e)} \in \bigtimes_{s\in N(e)}(Z_s)$ *with* $((q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)}) \in \Delta_e$*. An activated elementary automaton* (Φ_e, Δ_e) *can execute a state transition and produce a* successor state $p = (p_s)_{s\in\mathbb{S}} \in \bigtimes_{s\in\mathbb{S}}(Z_s)$*, if* $q_r = p_r$ *for* $r \in \mathbb{S} \setminus N(e)$ *and* $(q_s)_{s\in N(e)}, i, (p_s)_{s\in N(e)} \in \Delta_e$*. The corresponding state transition is* $(q, (e, i), p)$*.*

A state transition in the given model could for example cause a change in the state component *A_plvl_state* from (attacker,root).(db_server,none). . . into (attacker,root)(db_server,root). . . .

Attacker behaviour. Attacker capabilities are modelled by the atomic exploits and by the strategy to select and apply them.

A state transition modelling an exploit is constructed from, (1) a predicate that states that the attacker *knows* this exploit, (2) an expression to select source and target hosts for the exploit, (3) a predicate that states that the target *host is vulnerable* by this exploit, (4) an expression for the impact of the execution of this exploit on the attacker and on the target host as for example the shut down of services. Optional add-ons are, an assignment of cost benefit ratings to this exploit and intrusion detection checks.

Several different attackers can easily be included because an attacker is modelled as a role not a single instance and the tool can automatically generate multiple instances from one role definition.

Modelling of Denial of Service (DoS) attacks aiming to block resources or communication channels either directly or by side effects require a much more detailed model of the resources involved. This could be accomplished using the presented framework but is out of scope of this paper.

Some experiments have been made to generate a set of known exploits for the attacker(s) from a given algorithm. If for example it is assumed that the attacker knows 3 different exploits, then all combinations of 3 exploits from the set of all specified exploits have to be computed and further analysed. Another example for an attacker strategy is, that the attacker uses only exploits for vulnerabilities with a severity above a given threshold. This is based on the assumption, that the vulnerability severity reflects the probability of exploitation of a vulnerability.

Composition of a model and computation of an attack graph. The SH verification tool [4] is used to analyse this model. It manages the components of the model, allows to select alternative parts of the specification and automatically "glues" together the selected components to generate a combined model of ICT network specification, vulnerability and exploit specification, network security policy and attacker specification. After an initial

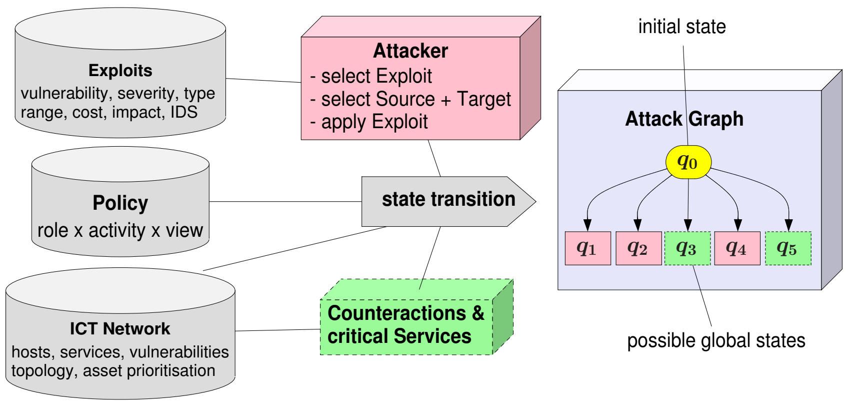


Fig. 4. Computation of the attack graph

configuration is selected, the attack graph (reachability graph) is automatically computed by the SH verification tool (see Fig. 4). Formally, the attack graph is the reachability graph of the corresponding APA model.

Definition 3. *The behaviour of an APA is represented by all possible coherent sequences of state transitions starting with initial state* q_0*. The sequence* $(q_0,(e_1,i_1),q_1)$ $(q_1,(e_2,i_2),q_2)$ $(q_2,(e_3,i_3),q_3)\ldots$ $(q_{n-1},(e_n,i_n),q_n)$ *with* $i_k \in \Phi_{e_k}$ *represents one possible sequence of actions of an APA.* q_n *is called the* goal *of this action sequence.*

State transitions $(p,(e,i),q)$ *may be interpreted as labelled edges of a directed graph whose nodes are the states of an APA:* $(p,(e,i),q)$ *is the edge leading from* p *to* q *and labelled by* (e,i)*. The subgraph reachable from the node* q_0 *is called the* reachability graph *of an APA.*

Let $\mathbb{Q}$ *denote the set of all states* $q \in \bigtimes_{s\in\mathbb{S}}(Z_s)$ *that are reachable from the initial state* q_0 *and let* Ψ *denote the set of all state transitions with the first component in* $\mathbb{Q}$*.*

The set $L \subset \Psi^*$ *of all action sequences with initial state* q_0 *including the empty sequence* ϵ *denotes the* action language *of the corresponding APA. The action language is prefix closed. By definition* q_0 *is the goal of* ϵ*.*

Attack graph of the example scenario. The computed attack graph for the simple example scenario is shown in Fig. 5. This graph was computed under the assumption that the attacker knows all exploits. Even if we assume as a more realistic attacker behaviour, namely that the attacker will only exploit vulnerabilities with a severity level above a given minimum, then the graph is still far too big to inspect it manually. Figure 6 shows a detail of this attack graph. Please note that for better readability the interpretations are omitted at the edge labels. For example the edge $q_{13} \longrightarrow q_{38}$ depicts the application of an exploit where attacker A uses the *ssh* vulnerability CAN_2003_0693 and there is a second exploit (which is stealth, that means not detectable by intrusion detection systems) with the same state transition. The edge $q_{38} \longrightarrow q_{73}$ depicts an action of the system to restart the *ssh*

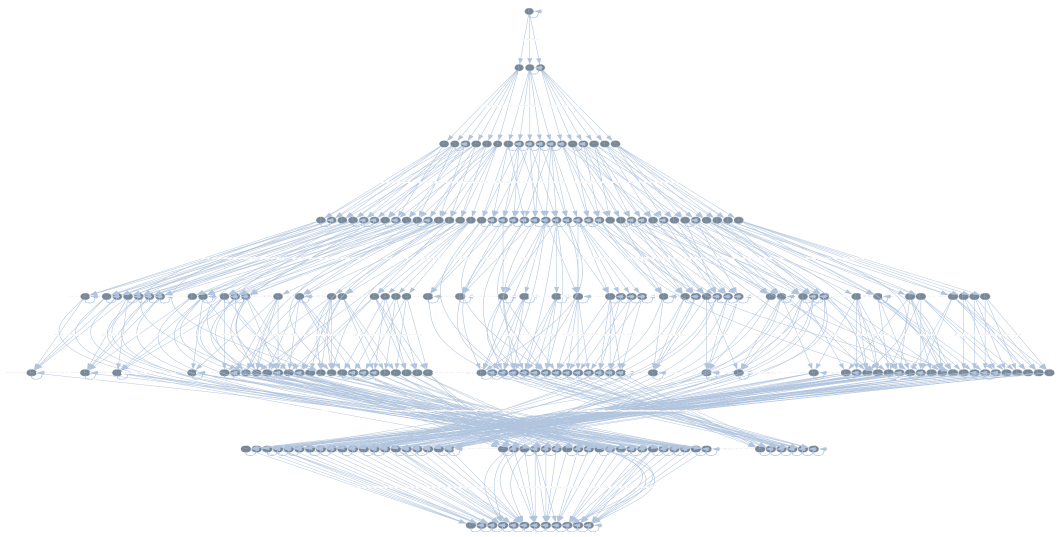


Fig. 5. Attack graph of simple example scenario

daemon and the edge $q_{73} \longrightarrow q_{73}$ depicts an action that models the availability of a critical service.

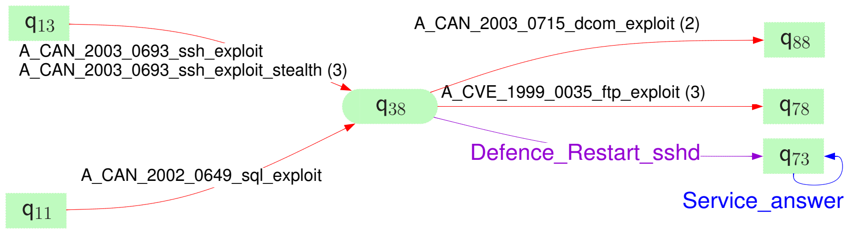

q13
A_CAN_2003_0693_ssh_exploit
A_CAN_2003_0693_ssh_exploit_stealth (3)
A_CAN_2003_0715_dcom_exploit (2)
q88
q38
A_CVE_1999_0035_ftp_exploit (3)
q78
Defence_Restart_sshd
q73
A_CAN_2002_0649_sql_exploit
Service_answer
q11

Fig. 6. Attack graph detail

For very large models, on the fly analysis allows to stop computation of the reachability graph automatically when specified conditions are reached or invariants are broken.

3 Evaluation of the model

3.1 Cost benefit analysis

Cost benefit analysis can be used as a means to help to assess the likely behaviour of an attacker. Cost ratings (from the view of an attacker) can be assigned to each exploit, for example to denote the time it takes for the attacker to execute the exploit or the resources needed to develop an exploit. Cost ratings can also be based on the severity ratings given by CVSS or US-CERT (cf. Sect.2.3). If not only technical vulnerabilities are modelled but also human weaknesses are considered, then cost could mean for example the money needed to *buy* a password.

Based on these cost assignments (weights of edges), the shortest path from the root of the attack graph to a node representing a successful attack can be computed using Dijkstra's well-known algorithm. This path represents the least expensive combined attack breaking a given security goal.

A benefit for the attacker based on the negative impact he achieves can also be assigned, for example to indicate the *worth* regarding relative criticality of the captured asset. Summarised costs and benefits can be compared for selected paths or the whole graph and used for example to find the node with the greatest benefit for a potential attacker. Please refer to [5] for an example.

Other security related properties such as the probability of being detected by intrusion detection systems can be associated with APA transitions. This information when evaluated in the analysis of an attack graph can lead to improvements of a given configuration of a critical information infrastructure.

3.2 Abstraction-based analysis

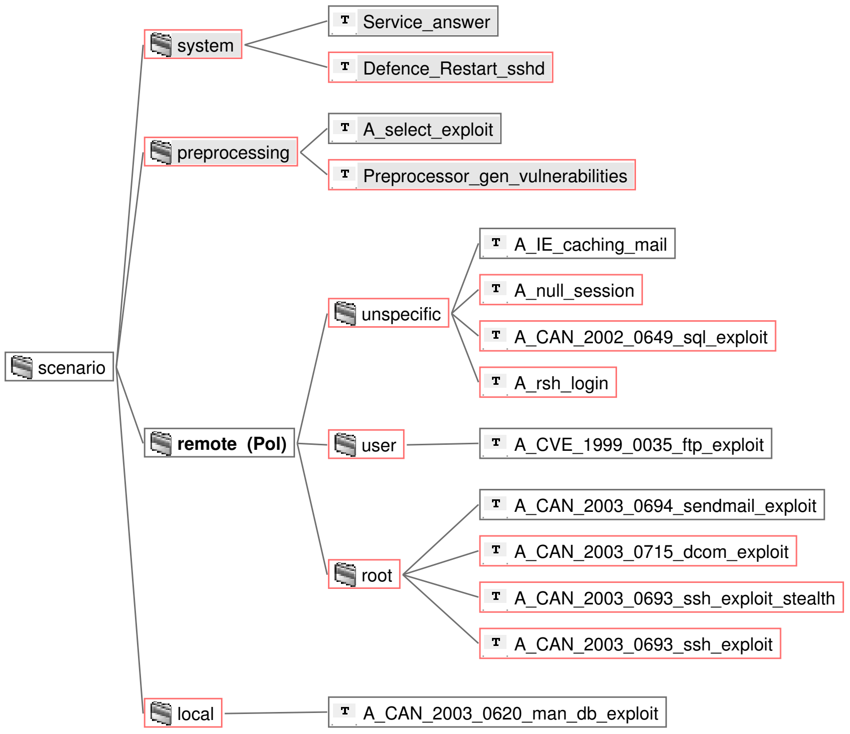
Abstract representations. The SH verification tool usually computes graphs of about 1 million edges in acceptable time and space. The problem now is, that it is impossible to visualise a graph of that size. However abstract representations of an attack graph can be computed and used to visualise and analyse compacted information focussed on interesting aspects of the behaviour.

Behaviour abstraction of an APA can be formalised by language homomorphisms, more precisely by *alphabetic language homomorphisms* $h : \Sigma^* \to \Sigma'^*$ on the action language.

By these homomorphisms certain transitions are ignored and others are renamed, which may have the effect, that different transitions are identified with one another. A mapping $h : \Sigma^* \to \Sigma'^*$ is called a *language homomorphism* if $h(\epsilon) = \epsilon$ and $h(yz) = h(y)h(z)$ for each $y, z \in \Sigma^*$. It is called *alphabetic*, if $h(\Sigma) \subset \Sigma' \cup \{\epsilon\}$.

The mappings used to compute the abstract representations of the behaviour have to be *property preserving*, to assure that properties are *transported* as desired from a lower to a higher level of abstraction and no critical behaviour is hidden by the mapping. Such properties, namely *simplicity*, are given in [6] and a check for simplicity is implemented in the SH verification tool [4]. The tool provides an editor to define homomorphisms on action languages, it computes corresponding minimal automata [7] for the homomorphic images and checks simplicity of the homomorphisms.

Example of a mapping to define an abstract representation. Figure 7 defines a mapping of the transitions representing an exploit of a vulnerability to the respective *range* and *impact type* assessments of the vulnerabilities as provided by NIST (cf. Sect. 2.3). Range types of the vulnerabilities in the example scenario are *remote* (remotely exploitable) and *local* (locally exploitable). Impact types used here are $unspecific$ (provides unauthorised access), *user* (provides user account access) and *root* (provides administrator access).



scenario
system
Service_answer
Defence_Restart_sshd
preprocessing
A_select_exploit
Preprocessor_gen_vulnerabilities
remote (Pol)
unspecific
A_IE_caching_mail
A_null_session
A_CAN_2002_0649_sql_exploit
A_rsh_login
user
A_CVE_1999_0035_ftp_exploit
root
A_CAN_2003_0694_sendmail_exploit
A_CAN_2003_0715_dcom_exploit
A_CAN_2003_0693_ssh_exploit_stealth
A_CAN_2003_0693_ssh_exploit
local
A_CAN_2003_0620_man_db_exploit

Fig. 7. Definition of an abstract representation of the attack graph

This mapping denotes, that all transitions (the leaves of the tree) are to be represented by their respective father nodes, namely *system*, *preprocessing*, *unspecific*, *user*, *root* and *local* in the abstract representation. The nodes *system* and *preprocessing* are coloured in grey, symbolising that they are mapped to ϵ, that means the transitions represented by these nodes will be invisible in the abstract representation. Please ignore the notation (Pol) at the node *remote* for the moment.

Figure 8 shows the result of application of the mapping in Fig. 7 to the attack graph from Fig. 5. This computed abstract representation (a graph with only 20 states and 37 edges) gives a visualisation focussing on the transition types *root*, *user*, *unspecific* and *local*. The simplicity of this mapping that guarantees that properties are preserved was automatically proven by the tool.

Refined mapping. To find out which policies permit the attacks shown in Fig. 8, a refinement of the abstraction defined in Fig. 7 is necessary. It is possible to "fine tune" the mapping so that the interpretation variables (cf. Sect. 2.5) stay visible in the abstract representation. In this case the binding of the interpretation variable Pol that contains the respective policy can be visualised. This is denoted by (Pol) in the node *remote* in Fig. 7. The corresponding refined abstract representation is a graph with 34 states and 121 edges when computed on the attack graph in Fig. 5. The initial nodes and edges of this graph

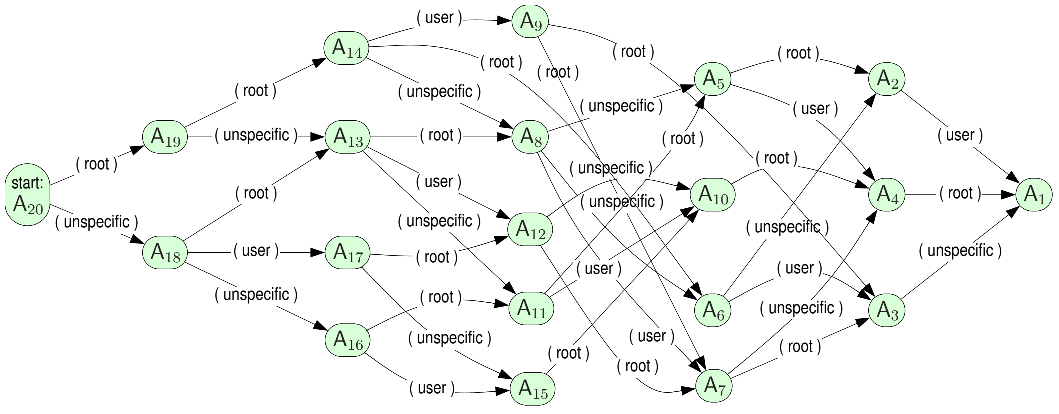


Fig. 8. Abstract view on an attack graph

are shown in Fig. 9(a). In comparison to the corresponding edges $A_{20} \longrightarrow A_{19}$ and $A_{20} \longrightarrow$

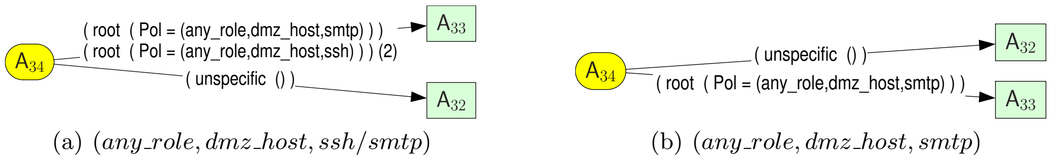

A34
(root (Pol = (any_role,dmz_host,smtp)))
(root (Pol = (any_role,dmz_host,ssh))) (2)
(unspecific ())
A33
A32
(a) $(any\_role, dmz\_host, ssh/smtp)$
A34
(unspecific ())
(root (Pol = (any_role,dmz_host,smtp)))
A32
A33
(b) $(any\_role, dmz\_host, smtp)$

Fig. 9. Details in the abstract view

A_{18} of the graph in Fig. 8 now the details on the related policies are visible.

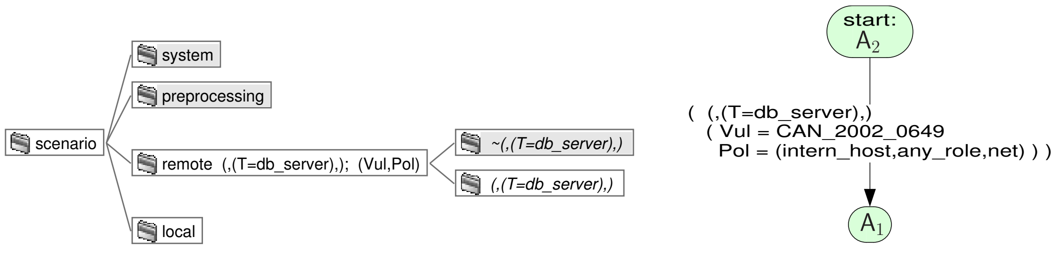
Adapt/optimise the system configuration. Further analysis reveals, that, if the example policy given in Fig. 3(b) is changed to allow only *smtp* instead of *ssh* and *smtp* for *any_role* to *dmz_host* then the analysis yields of course a smaller graph than the original shown in Fig.5, the coarse abstract representation in Fig. 8 is the same, but the finer mapping with interpretation variable *Pol* visible results in a different representation which is shown in Fig. 9(b).

If alternatively the policy is restricted to allow only *ssh* instead of *ssh* and *smtp* in the above example, then the result is yet a different attack graph but the abstract view in Fig. 8 is still the same.

This analysis demonstrates that there may be differences in the detailed attack graphs but no differences in the abstract representations thereof. This indicates that the different policies are equally effective (or not) concerning the enforcement of security goals on the abstract level, even if variations in the attack paths are covered by different policy rules.

Using predicates to define abstractions. Let us now assume that the host *db_server* in the scenario is the most valuable and mission critical host in the ICT network. So we want to know if in the given scenario, (1) attacks to the db_server are possible, (2) on which vulnerabilities they are based, and, (3) what policy rules are directly involved.

The abstraction in Fig. 10(a) exemplifies how predicates can be used to define such a mapping. In this mapping the predicate (T=db_server) matches only those transitions that model direct attacks to the target host db_server. Furthermore the bindings of the interpretation variables Vul and Pol that contain the respective vulnerability and policy are used in the mapping. The remote transitions that don't match that predicate are mapped to ϵ and so are invisible.



(a) Abstraction used (b) Resulting graph

Fig. 10. Focus on attacks to the host *db_server*

Evaluating this abstraction on the attack graph from Fig. 5 above results in the simple graph given in Fig. 10(b). This proves that, (1) in the current policy configuration attacks to the db_server are possible, (2) those attacks are based on exploits of the vulnerability CAN_2002_0649, and, (3) they are utilising the policy permission (intern_hosts,any_role,net). So to prevent this attack, it has to be decided, whether it is more appropriate to uninstall the product that is hurt by this vulnerability or to restrict the internal hosts in their possible actions by replacing the above policy with a more restrictive one.

3.3 Liveness properties

As it is well known, system properties are divided into two types: safety (what happens is not wrong) and liveness properties (eventually something desired happens) [8]. Liveness properties in the context of ICT infrastructure security analysis cover availability and business continuity aspects for example with respect to denial of service attacks.

On account of liveness aspects system properties are formalised by ω-languages (sets of infinite long words). So to investigate satisfaction of properties "infinite system behaviour" has to be considered. This is formalised by so called Eilenberg limits of action languages (more precisely: by Eilenberg limits of modified action languages where maximal words are continued by an unbounded repetition of a dummy action) [9].

The usual concept of linear satisfaction of properties (each infinite run of the system satisfies the property) is not suitable in this context because no fairness constraints are considered. We put a very abstract notion of fairness into the satisfaction relation for properties, which considers that independent of a finitely long prefix computation of a system certain desired events may occur eventually. To formalise such "possibility properties", which are of interest when considering what we call cooperating systems, the notion of approximate satisfaction of properties is defined in [9].

Definition 4. *A system* approximately satisfies *a property if and only if each finite behaviour can be continued to an infinite behaviour, which satisfies the property.*

For safety properties linear satisfaction and approximate satisfaction are equivalent [9]. To deduce approximately satisfied properties of a specification from properties of its abstract behaviour an additional property of abstractions called simplicity of homomorphisms on an action language [10] is required. Simplicity of homomorphisms is a very technical condition concerning the possible continuations of finite behaviours.

For regular languages simplicity is decidable. In [10] a sufficient condition based on the strongly connected components of corresponding automata is given, which easily can be checked. Especially: If the automaton or reachability graph is strongly connected, then each homomorphism is simple. The following theorem [9] shows that approximate satisfaction of properties and simplicity of homomorphisms exactly fit together for verifying cooperating systems.

Theorem 1. Simple homomorphisms *define exactly the class of such abstractions, for which holds that each property is approximately satisfied by the abstract behaviour if and only if the "corresponding" property is approximately satisfied by the concrete behaviour of the system.*

Formally, the "corresponding" property is expressed by the inverse image of the abstract property with respect to the homomorphism.

When a system's countermeasures and the behaviour of vital services the system provides are included in the model, then availability properties such as the system's resilience with respect to denial of service attacks can be analysed.

The state transitions $Defence\_restart\_sshd$ and $Service\_answer$ in Fig. 6 give an example for a modelling of system countermeasures and critical services availability. If for example as a side effect of an *ssh_exploit* the attacker kills the *sshd* then afterwards the *sshd* is not active on the respective host and so some service possibly cannot answer requests anymore. Now additionally a system countermeasure is considered that restarts the *sshd*. No other details are added to keep the model small. A typical liveness question in this scenario is "Will a client still get answers from a server when the network is attacked ?". Using the appropriate type of model checking, *approximate satisfaction* of temporal logic formulae can be checked by the SH verification tool [11], [4]. In terms of temporal logic the property in question above can be written as $G\ F\ Service\_answer$ (always eventually $Service\_answer$) which is found to be *true* by the tool.

4 Resilience against exploits of unknown vulnerabilities

One way to consider resilience of an information infrastructure against attacks to unknown vulnerabilities is, to define a new vulnerability for each installed product. For the model of the scenario used in this paper this has been done by definition of a new vulnerability called CAN_generic with a variable part for the affected service. In the same way a generic exploit based on this vulnerability is defined. Now in the preprocessing phase a state transition selects an arbitrary product and inserts a generic vulnerability CAN_generic for that product and the related service. Because the reachability analysis considers every possible choice of product, all alternatives are evaluated in the attack graph.

When analysing the (now much larger) attack graph, the mapping in Fig. 11 exemplarily shows a possible use of resilience analysis. The state transition modelling an exploit of an unknown generic vulnerability uses the additional interpretation variables RS and RT, where RS denotes the role of the source host and RT the role of the target host. So the given predicate ($RS = RT$) matches only those transitions that model attacks of hosts in the same role (within the same zone). Now the attacks that fulfil this predicate are mapped to ϵ (coloured in grey in the mapping) and so are invisible, whereas the attacks with $RS \neq RT$ (across roles/zones) are visible. Furthermore the bindings of the interpretation variables $VulServ$ and Pol that contain the respective vulnerable service and policy are used in the mapping. All other transitions are mapped to ϵ.

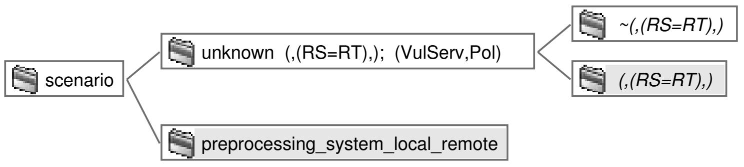

scenario
unknown (,(RS=RT),); (VulServ,Pol)
~(,(RS=RT),)
(,(RS=RT),)
preprocessing_system_local_remote

Fig. 11. Mapping for attacks against unknown vulnerabilities that cross zones

The abstract representation computed from that mapping is shown in Fig. 12. It gives a clear overview about what kind of zone crossing attacks would be possible in case that new unknown vulnerabilities were exploited. For each assumed vulnerable service it shows the policies that would allow the attack.

Using a modified definition of the mapping in Fig. 11 allows to look into more detail for example behind the edge ($VulServ = sendmaild\ Pol = (intern\_host,\ any\_role, net)$) shown in bold font in Fig. 12. A refined mapping with predicate ($RS \neq RT\ \wedge\ VulServ = sendmaild$) and visible interpretation variables RS and RT results in an abstract representation with 4 parallel edges labelled ($RS = developer\_host\ RT = dmz\_host$), ($RS = db\_host\ RT = dmz\_host$), ($RS = intern\_host\ RT = dmz\_host$) and ($RS = management\_host\ RT = dmz\_host$) respectively. This shows that if an attacker knows a new exploit for an unknown vulnerability of the product providing the *sendmaild*, then the current policy rule ($intern\_host, any\_role, net$) would allow to use the exploit to cross the 4 given zones.

Now if the policies are quite restrictive and no new cross role/zone attacks are found by the reachability analysis, then it can be concluded that the network configuration is resilient

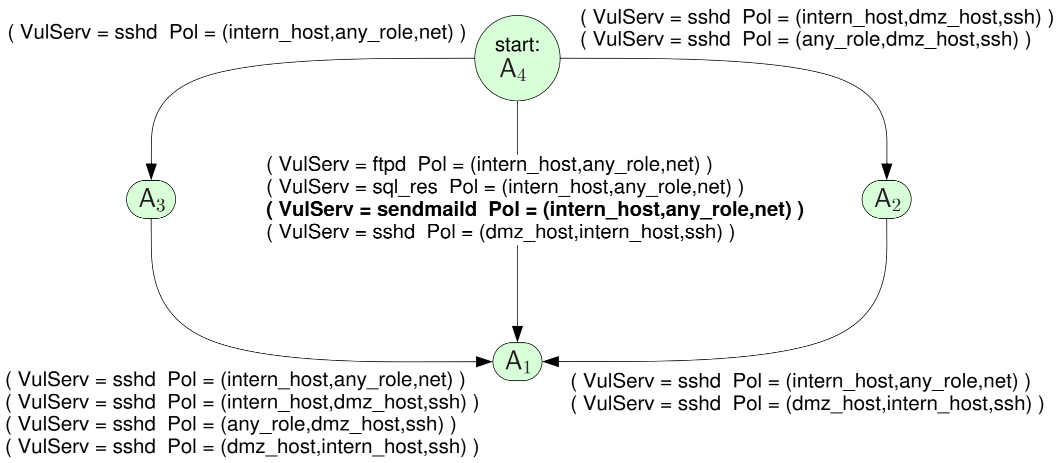

start:
A_4
A_3
A_2
A_1
(VulServ = sshd Pol = (intern_host,any_role,net))
(VulServ = sshd Pol = (intern_host,dmz_host,ssh))
(VulServ = sshd Pol = (any_role,dmz_host,ssh))
(VulServ = ftpd Pol = (intern_host,any_role,net))
(VulServ = sql_res Pol = (intern_host,any_role,net))
(VulServ = sendmaild Pol = (intern_host,any_role,net))
(VulServ = sshd Pol = (dmz_host,intern_host,ssh))
(VulServ = sshd Pol = (intern_host,any_role,net))
(VulServ = sshd Pol = (intern_host,dmz_host,ssh))
(VulServ = sshd Pol = (any_role,dmz_host,ssh))
(VulServ = sshd Pol = (dmz_host,intern_host,ssh))
(VulServ = sshd Pol = (intern_host,any_role,net))
(VulServ = sshd Pol = (dmz_host,intern_host,ssh))

Fig. 12. Abstract representation of attacks against unknown vulnerabilities

with respect to attacks against *one* unknown vulnerability. In the same way resiliency with respect to two or more unknown vulnerabilities can be analysed. Please note that in many cases this will not be possible because of state space explosion problems but computation of a section of the attack graph by giving a limitation on the number of edges to be computed is possible and should help to find problems and to successively restrict the configuration to an acceptable risk level.

5 Related work

This paper is based on the work presented in [12]. The network vulnerability modelling part of the framework presented here is adopted from the approach introduced in [5] and is similar in design to an approach by Phillips and Swiler in [13] and [14]. Related work of Jha, Sheyner, Wing et al. used attack graphs that are computed and analysed based on model checking in [15] and [16]. Ammann et al. presented an approach in [17] that is focussed on reduction of complexity of the analysis problem by explicit assumptions of monotonicity.

To seamlessly integrate the methods and tool presented here into a network vulnerability analysis framework, a tool-assisted transformation of up-to-date ICT system configuration and vulnerability databases into a formal specification of the model is required. This should preferably be based on automatically updated information of network scanners because administration databases are typically out-of-date. Recent work by Noel, Jajodia et al. in [18] and [19] already covers this aspect and also describes attack graph visualisation techniques.

The work of Kotenko and Stepashkin in [20] is focussed on security metrics computations and adaptive cooperative defence mechanisms [21].

To model the ICT network, the vulnerabilities and the intrusion detection systems, a data model loosely resembling the formally defined M2D2 information model [1] is used. Appropriate parts of this model are adopted and supplemented by concepts needed for description of exploits, attacker knowledge and strategy and information for cost benefit analysis. A formal approach to use an Organisation-Based Access Control (Or-BAC) model to specify network security policies was presented in [2]. This approach is adopted here to model the network security policies in the attack graph analysis framework.

The modelling framework is based on *Asynchronous Product Automata (APA)*, a flexible operational specification concept for cooperating systems [11]. The applied analysis method is implemented in the SH verification tool [4] that has been adapted and extended to support the presented attack graph evaluation methods.

Major focus of the combined modelling framework presented in this paper, is the integration of formal network vulnerability modelling on the one hand and network security policy modelling on the other hand. This aims to help adaptation of a network security policy to a given and possibly changing vulnerability setting. Recent methods for analysis of attack graphs are extended to support analysis of abstract representations of these graphs.

Extensions to the APA-based model checking techniques are needed to be able to verify entire families of systems, independent of the exact number of replicated components. Such an approach to abstraction-based analysis of parameterised policy controlled systems is presented in [22].

6 Further research objectives

The work presented in this paper brings together, (1) attack graph computation technology, (2) state-of-the-art policy modelling, and, (3) formal methods for analysis and computation of abstract representations of the system behaviour. The aim is, to guide a systematic evaluation and assist the persons in charge with optimising adaptation of the network security policy to an ever-changing vulnerability setting and so to improve the configuration of the information infrastructure.

(Security) metrics in abstract representations. A summarisation of severity ratings for single security vulnerabilities as provided by CVSS or US-CERT (cf. Sect. 2.3) based on attack graphs has been addressed in recent work of Kotenko and Stepashkin [20]. Interesting questions in such an approach are, which attacker strategy or bundle of strategies to apply and how to "condense" the information in the graph into a comprehensive measure of the security of an ICT network.

In an abstraction-based approach a method to assign measures to the nodes or edges in the abstract representations is needed. One idea is to look at the origin nodes (nodes in the attack graph) of an abstract node and then for example compute the minimum value of some measure from the set of origin nodes. If for example a shortest path analysis (cf. Sect.3.1) was computed on the attack graph, then each node of the attack graph is associated with a value for the shortest (least expensive) path to that node. If now the semantics of these values allows a comparison, then a function such as the minimum measure from the set of origin nodes can be associated with each node in the abstract representation. If the cost ratings of the transitions in the attack graph are based on severity ratings from CVSS or US-CERT (cf. Sect.2.3) then the function for the transformation of the values in the origin nodes to the abstract representation can be used for a metric of security of the critical information under the abstract view defined by the mapping. Consideration of *resilience* *against exploits of unknown vulnerabilities* (cf. Sect. 4) could also contribute to such a measure.

Threat response mechanisms. An even more advanced objective is, to extend this framework to support *policy-based, automated threat response* that makes use of alert information. Such a self-adaptive response mechanism could substantially improve the resilience of policy controlled ICT systems against network attacks. A framework for simulation of adaptive cooperative defense against internet attacks has been presented by Kotenko and Ulanov in [21]. Analysis of distributed coordinated attacks and the controlling mechanisms such as botnets using the abstraction-based approach presented in this paper would complement their approach.

Acknowledgements. The author likes to thank Peter Ochsenschläger who conducted most of the theoretical work on APA that this work is based on and Jürgen Repp who implemented most of the SH verification tool that was used as basis for the modelling and analysis presented here.

Part of the work presented in this paper was developed within the project SicAri being funded by the German Ministry of Education and Research.

References

1. Morin, B., Mé, L., Debar, H., Ducassé, M.: M2d2: A formal data model for ids alert correlation. In: Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002, Zurich, Switzerland, October 16-18, 2002, Proceedings. Volume 2516 of Lecture Notes in Computer Science., Springer (2002) 115–137
2. Cuppens, F., Cuppens-Boulahia, N., Sans, T., Miège, A.: A formal approach to specify and deploy a network security policy. In: Second Workshop on Formal Aspects in Security and Trust (FAST). (2004)
3. Schiffmann, M.: A Complete Guide to the Common Vulnerability Scoring System (CVSS) (2005) http://www.first.org/cvss/cvss-guide.html.
4. Ochsenschläger, P., Repp, J., Rieke, R.: The SH-Verification Tool. In: Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000), Orlando, FL, USA, AAAI Press (2000) 18–22
5. Rieke, R.: Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration. In: In U.E. Gattiker (Ed.), Eicar 2004 Conference CD-rom: Best Paper Proceedings, Copenhagen, EICAR e.V. (2004)
6. Ochsenschläger, P., Repp, J., Rieke, R.: Verification of Cooperating Systems – An Approach Based on Formal Languages. In: Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS-2000), Orlando, FL, USA, AAAI Press (2000) 346–350
7. Eilenberg, S.: Automata, Languages and Machines. Volume A. Academic Press, New York (1974)
8. Alpern, B., Schneider, F.B.: Defining liveness. Information Processing Letters **21**(4) (1985) 181–185
9. Nitsche, U., Ochsenschläger, P.: Approximately satisfied properties of systems and simple language homomorphisms. Information Processing Letters **60** (1996) 201–206

10. Ochsenschläger, P.: Verification of cooperating systems by simple homomorphisms using the product net machine. In Desel, J., Oberweis, A., Reisig, W., eds.: Workshop: Algorithmen und Werkzeuge für Petrinetze, Humboldt Universität Berlin (1994) 48–53
11. Ochsenschläger, P., Repp, J., Rieke, R., Nitsche, U.: The SH-Verification Tool Abstraction-Based Verification of Co-operating Systems. Formal Aspects of Computing, The International Journal of Formal Method **11** (1999) 1–24
12. Rieke, R.: Modelling and Analysing Network Security Policies in a Given Vulnerability Setting. In: Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos Island, Greece. Volume 4347 of LNCS., Springer (2006) 67–78 © Springer.
13. Phillips, C.A., Swiler, L.P.: A graph-based system for network-vulnerability analysis. In: NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms, September 22-25, 1998, Charlottsville, VA, USA, ACM Press (1998) 71–79
14. Swiler, L.P., Phillips, C., Ellis, D., Chakerian, S.: Computer-attack graph generation tool. In: DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume 2,June 12 - 14, 2001, Anaheim, California, IEEE Computer Society (2001) 1307–1321
15. Jha, S., Sheyner, O., Wing, J.M.: Two formal analyses of attack graphs. In: 15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada, IEEE Computer Society (2002) 49–63
16. Sheyner, O., Haines, J.W., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: 2002 IEEE Symposium on Security and Privacy, May 12-15, 2002, Berkeley, California, USA, IEEE Comp. Soc. Press (2002) 273–284
17. Ammann, P., Wijesekera, D., Kaushik, S.: Scalable, graph-based network vulnerability analysis. In: Proceedings of the 9th ACM conference on Computer and communications security, ACM Press New York, NY, USA (2002) 217–224
18. Noel, S., Jajodia, S.: Managing attack graph complexity through visual hierarchical aggregation. In: VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security, New York, NY, USA, ACM Press (2004) 109–118
19. Noel, S., Jacobs, M., Kalapa, P., Jajodia, S.: Multiple Coordinated Views for Network Attack Graphs. In: IEEE Workshop on Visualization for Computer Security (VizSec'05), Los Alamitos, CA, USA, IEEE Computer Society (2005)
20. Kotenko, I., Stepashkin, M.: Analyzing Network Security using Malefactor Action Graphs. International Journal of Computer Science and Network Security **6** (2006)
21. Kotenko, I., Ulanov, A.: Multi-agent Framework for Simulation of Adaptive Cooperative Defense against Internet Attacks. In: In Proceedings of International Workshop on Autonomous Intelligent Systems: Agents and Data Mining (AIS-ADM-07). Lecture Notes in Artificial Intelligence, Vol.4476. (2007)
22. Ochsenschläger, P., Rieke, R.: Abstraction Based Verification of a Parameterised Policy Controlled System. In: International Conference "Mathematical Methods, Models and Architectures for Computer Networks Security" (MMM-ACNS-07). Volume 1 of CCIS., Springer (2007) © Springer.