Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration

Roland Rieke
Fraunhofer Institute for Secure Telecooperation, Germany

About the author

Born in 1957 in Braunschweig, Roland Rieke studied computer science from 1976 to 1982 at TUD (Darmstadt University of Technology) and has worked as a researcher for SIT (the Fraunhofer Institute Secure Telecooperation, former GMD) in Darmstadt since 1982. He is married and has 3 children. Projects he worked on comprise the design and implementation of components for distributed office systems and network protocols. Since 1996 his research interests are focused on the development of methods and tools for formal security models and application of these techniques. Recently he has worked on the proof of security characteristics of cryptographic protocols, the java virtual machine verifier, context oriented security policies and e-government applications.

Mailing Address: Fraunhofer Institut Sichere Telekooperation, Rheinstrasse 75, D-64295 Darmstadt, Germany; Phone: +49 6151869284; Fax: +49 6151869224; Email: rieke@sit.fraunhofer.de

Descriptors
critical infrastructure protection, attack simulation, verification tool, security properties, survivability analysis, cost-benefit analysis, intrusion detection, countermeasure evaluation, critical services, risk assessment

Reference to, or **Citation** of this paper should be made as follows:
Rieke, R. (2004). Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration. In U.E. Gattiker (Ed.), EICAR 2004 Conference CD-rom: Best Paper Proceedings (ISBN: 87-987271-6-8) 31 pages. Copenhagen: EICAR e.V.

Tool based formal Modelling, Analysis and Visualisation of Enterprise Network Vulnerabilities utilising Attack Graph Exploration

Abstract

A core concern of critical infrastructure protection is a careful analysis of what parts of the information infrastructure really need protection and what are the concrete threads as well as an evaluation of appropriate protection measures. In this paper a methodology and a tool for the development and analysis of operational formal models is presented that addresses these issues in the context of network vulnerability analysis.

A graph of all possible attack paths is automatically computed from the model of a government or enterprise network, of vulnerabilities, exploits and an attacker strategy.

Based on this graph, security properties are specified and verified, abstractions of the graph are computed to visualise and analyse compacted information focussed on interesting aspects of the behaviour and cost-benefit analysis is performed.

Survivability comes into play, when systems' countermeasures and the behaviour of vital services it provides are also modelled and effects are analysed.

Introduction

Today's public, government and enterprise networks are facing an accumulation of risks because a multitude of more or less critical vulnerabilities to system security are found every month. At the same time, the published malicious incidents increase in scope and severity. On the other hand, technological advancements in antivirus software, firewalls and intrusion detection systems provide a broad palette of proactive defence measures for network protection and impact reduction. The increasing complexity of the network structures and possible protection strategies on one hand and the attack possibilities on the other hand require tool based methods, to guide a systematic evaluation and assist the persons in charge with finally determining exactly what really needs protection and which strategy and means to apply.

A typical means by which an attacker tries to break into a network is, to use combinations of basic exploits to get more information or more credentials and to capture more hosts 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*. It is also important, to find out which protection could block successful attack paths most efficiently or at least detect attack attempts in an early phase.

For this type of vulnerability analysis, an operational formal model is presented that represents the information system and the behaviour of an attacker. In more detail, it comprises a model of the enterprise network structure and configuration including intrusion detection components, a model of vulnerabilities and corresponding basic exploits, a model of attacker capabilities and profile, and optionally a model of the system's countermeasures.

Based on that model, a reachability graph representing the complete system behaviour is automatically computed. Because this graph in the presented application scenario represents all possible attack paths, it is called *attack graph* in the following text. Now security properties can be specified and verified on the computed behaviour of the model.

The applied verification method is based on formal methods and is implemented in the *SH verification tool* (Ochsenschläger et al., 1999, 2000a) that has been adapted and extended to support the presented attack graph analysis methods. Questions relating to security properties that can be answered by analysing the attack graph include the following:

Q 1 What *security goals* can be *broken* by a combination of a set of basic exploits selected as attacker profile ?

Q 2 Find the biggest *sources of trouble* in the system based on vulnerability-priorities network-structure and possible attack- patterns. Is there a *critical host or vulnerability* on all paths to some attacker goal ?

Q 3 Quick check of "*am I affected*" by a newly found vulnerability and what new attackcombinations/patterns are possible when adding this vulnerability ?

Q 4 What are the *effects of changes* to the network configuration on overall vulnerability ?

If the model additionally includes specifications of intrusion detection components, then their behaviour and required coaction to recognise attacks, even when evidence is scattered over several hosts, can be analysed.

Common questions concerning intrusion detection are:

Q 5 What *attacks* are *detected* ?

Q 6 What are the *effects of changes* to intrusion detection systems on overall detection of attacks ?

Abstractions of the attack graph can be computed to visualise and analyse compacted information focussed on interesting aspects of the behaviour. The mappings used to compute the abstracted 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.

Aspects that can be visualised using appropriate abstractions on the attack graph are for example:

Q 7 How does the attack graph look like when only attacks that give the attacker new root access are shown (focussing on *gain of credentials*) ?

Cost-benefit analysis can be performed based on costs assigned to the atomic exploits representing level of effort for the attacker and benefits regarding relative importance of the captured hosts. Typical questions concerning cost-benefit are:

Q 8 What is the attack with the *least costs* breaking a given security property ?

Q 9 How much *impact* can an attacker produce given that he applies a given set of atomic exploits ?

Q 10 What is the *optimal position* of given intrusion detection systems regarding cost benefit balance ?

Liveness (in this context often called survivability) comes into play, if part of the behaviour of the enterprise network is also modelled. Analysing effects of countermeasures the system performs under attack or the behaviour of vital services it provides is possible. Careful modelling on an adequate abstraction level is required here to avoid typical state space explosion problems. A typical liveness questions is:

Q 11 Is a *client* still able to *get answers* from a DBserver when the enterprise network is under attack ?

Some remarks on the remainder of this paper:
The first step in critical infrastructure protection is, to identify the organisation's critical infrastructures and to determine the threats against those infrastructures. This process is described in the next section particularly with regard to network vulnerability analysis. For this purpose, the components to be specified for modelling an attack scenario are described in detail.

The next step in critical infrastructure protection is, to analyse the vulnerabilities of the threatened infrastructure, to assess the risks of degradation or loss of a critical resources as well as to evaluate the effects of the application of countermeasures where risk is unacceptable. To support that process in the given context, in the subsequent section a methodology for the analysis of an attack graph is presented that helps to reveal complex attack combinations and supports the systematic evaluation of possible solutions to minimise risk with given resources.

In the last section an example scenario is presented and the dynamic behaviour of different variants is analysed. Finally some related work is commented, conclusions from this work are drawn and further research goals are sketched.

Modelling an attack scenario

In this section the information model used and the formal analysis and verification methods and the tool are described, the required specifications are explained in detail and the computation of the attack graph is outlined. Figure 1 shows an overview of the components used to specify the model of the enterprise network system under attack.

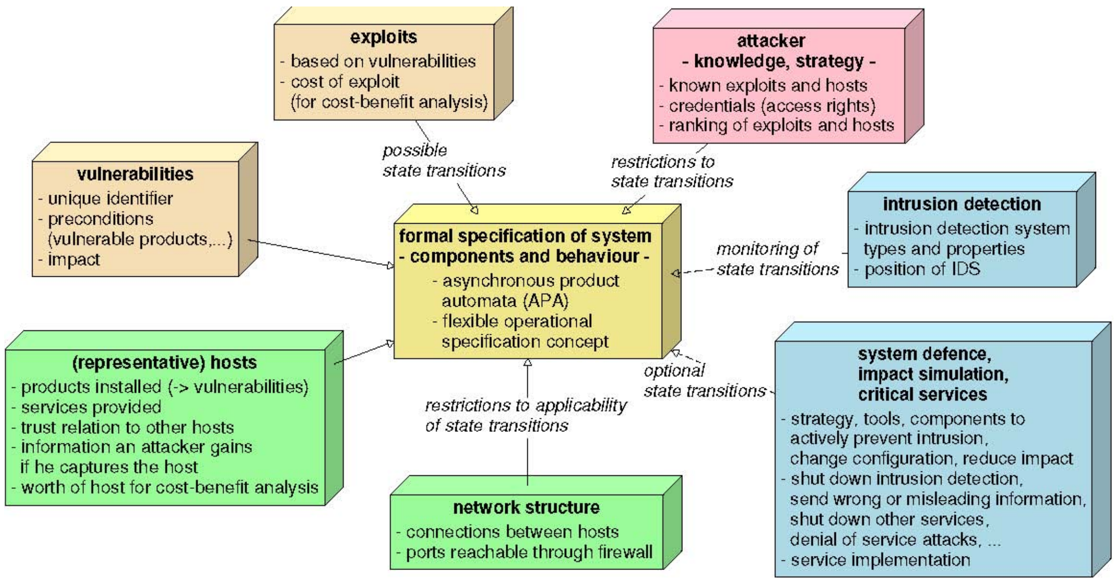

exploits
- based on vulnerabilities
- cost of exploit
(for cost-benefit analysis)
attacker
- knowledge, strategy -
- known exploits and hosts
- credentials (access rights)
- ranking of exploits and hosts
vulnerabilities
- unique identifier
- preconditions
(vulnerable products,...)
- impact
possible
state transitions
restrictions to
state transitions
formal specification of system
- components and behaviour -
- asynchronous product
automata (APA)
- flexible operational
specification concept
monitoring of
state transitions
intrusion detection
- intrusion detection system
types and properties
- position of IDS
(representative) hosts
- products installed (-> vulnerabilities)
- services provided
- trust relation to other hosts
- information an attacker gains
if he captures the host
- worth of host for cost-benefit analysis
restrictions to applicability
of state transitions
optional
state transitions
system defence,
impact simulation,
critical services
- strategy, tools, components to
actively prevent intrusion,
change configuration, reduce impact
- shut down intrusion detection,
send wrong or misleading information,
shut down other services,
denial of service attacks, ...
- service implementation
network structure
- connections between hosts
- ports reachable through firewall

Figure 1. Components of the model.

Information model

To model the enterprise network, the vulnerabilities and the intrusion detection systems, a data model loosely resembling the M2D2 information model (Morin et al., 2002) is used. M2D2 is a formally defined model for information related to the characteristics of the monitored information system, information about the vulnerabilities, information about the security tools used for the monitoring, and information about the events observed. 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.

Modelling hosts and network topology

The set of all hosts of the information system consists of the union of the hosts of the enterprise network and the hosts of the attacker(s).

A somewhat abstracted view is used for the representation of network topology including firewalls in the information model. A relation stating what port on what host is reachable from one another is used as network model. The model is very flexible, so that this implicit representation may be changed to a more explicit representation of firewalls easily if this turns out to be useful.

Modelling products, vulnerabilities and host configurations

Following the M2D2 model *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. Additionally to the installed products a host configuration contains information about what services are currently running and on what ports they are listening.

The vulnerabilities are represented in form of specifications representing a (sub)set of *common vulnerabilities and exposures CVE/CAN* that *MITRE* (see http://cve.mitre.org/) provides to support standardisation of names for all publicly known vulnerabilities and security exposures. These specifications additionally include preconditions about the target host as well as network preconditions and describe effects that the vulnerabilities have on the attacker and possibly on the network and target host.

Representative hosts

When analysing a complex enterprise network one usually faces a state space explosion problem because all possible combinations of exploits on all possible hosts have to be explored. Therefore it is advantageous to subsume all groups of hosts that have the same configuration, run the same products and are reachable with the same restrictions and that exhibit the same behaviour to one representative host for each such group. In the following text the term hos*t* will be used as a synonym referring to this representative host. What is suggested here, is to have an abstraction layer between the real enterprise network and the network of representative hosts that still contains all relevant attacks but reduces equivalent combinations. This abstraction could also be applied later after analysing the complete behaviour of the system by using an appropriate mapping but analysis takes much longer then because all sequences of possible combinations have to be computed.

Summarising representative hosts

An extension of the above sketched strategy (if the network is still too big for analysis) is to summarise hosts that are reachable with the same restrictions and add up their vulnerabilities to create a representative host with merged vulnerabilities of all summarised hosts. In this case some attacks may be found that are not possible in the real network and the decision if this approximation of system behaviour is good enough for analysis is up to the modeller. A strategy could be, to start with only one representative host per operating system that is configured to have installed all vulnerable products that the enterprise uses (for that operating system) and after that analysis go to a finer granularity as long as the computed state space is still manageable.

Automated generation of formal specifications ?

Note that it would be desirable to have an automated generation of formal specifications of system configuration directly derived from the output that network scanner tools like Nessus (see http://www.nessus.org/) provide.

Furthermore vulnerability specifications could be derived from vulnerability database information that for instance *ICAT* (see http://icat.nist.gov/) provides. First step would be to find a good structure and means for a formal description of vulnerabilities that can be used to collect a database of all known vulnerabilities. An agreed upon formal (and tool readable) description of intruder/host/service/networkpreconditions and effects of exploitation would have to be developed. An international project like the CAMDIER proposal (Gattiker et al., 2003) might tackle such a task.

Operational specification of the behaviour

The modelling of the behaviour of the given information model is based on *asynchronous product automata* (APA), a flexible operational specification concept for cooperating systems (Gürgens et al., 2002b). An APA consists of a family of so called *elementary automata* communicating by common components of their state (shared memory). APA are formally defined in figure 2.

Formally an *Asynchronous Product Automaton* 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* $N : \mathbb{E} \to \mathcal{P}(\mathbb{S})$; $\mathcal{P}(X)$ is the power set of X and $\mathbb{S}$ and $\mathbb{E}$ are index sets with the names of state components and elementary automata. For each Elementary Automaton (Φ_e, Δ_e) with *Alphabet* Φ_e, $\Delta_e \subseteq \times_{S \in N(e)}(Z_S) \times \Phi_e \times \times_{S \in N(e)}(Z_S)$ is its *State Transition Relation*. 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 $\times_{S \in \mathbb{S}}(Z_S)$. 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* $s_0 = (q_{0S})_{S \in \mathbb{S}} \in \times_{S \in \mathbb{S}}(Z_S)$. In total, an APA $\mathbb{A}$ is defined by $\mathbb{A} = ((Z_S)_{S \in \mathbb{S}}, (\Phi_e, \Delta_e)_{e \in \mathbb{E}}, N, s_0)$.
The behaviour of an APA is represented by all possible sequences of state transitions starting with initial state s_0. The sequence $(s_0, (e_1, a_1), s_1)(s_1, (e_2, a_2), s_2)(s_2, (e_3, a_3), s_3) \ldots$ with $a_i \in \Phi_{e_i}$ represents one possible sequence of actions of an APA.
State transitions $(s, (e, a), \bar{s})$ may be interpreted as labeled edges of a directed graph whose nodes are the states of an APA: $(s, (e, a), \bar{s})$ is the edge leading from s to $\bar{s}$ and labeled by (e, a). The subgraph reachable from the node s_0 is called the *reachability graph* of an APA.

Figure 2. APA definition.

APA state components representing the information model

The information model described above is specified for the proposed analysis method using the following *APA state components*:

S 1 a specification of the enterprise network topology and host configurations
reachability of ports on all hosts
- trust relations between hosts
- knowledge available at each host that might be valuable for an attacker as for example ipnumbers of other reachable hosts
- services running on each host
- installed products on each host

S 2 a specification of vulnerabilities of products

➔ leads to a specification of vulnerabilities for each host S 2' when combined with products installed on each host specified above.

S 3 a specification of attacker knowledge and strategy

S 4 a specification of installed intrusion detection components

S 5 cost benefit ratings, when evaluation about relative values is intended

These specifications are represented in the data structures and initial configuration of the state components in the APA model (see figures 3 and 4).

Modelling attacker and system behaviour

APA state transitions are used to represent atomic exploits and optionally actions the enterprise network system can take to defend itself or to implement vital services (see figures 3 and 4).

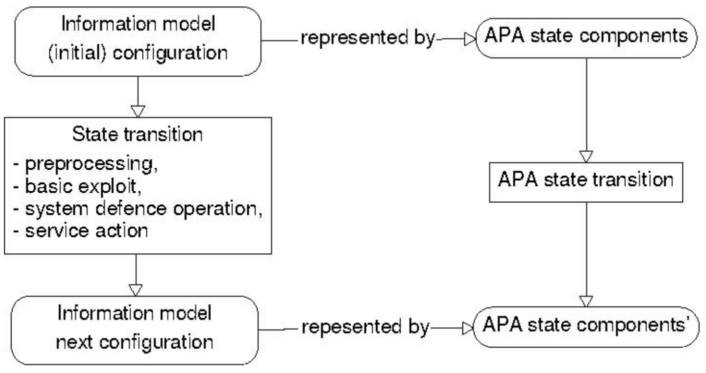


Figure 3. Representation of the information model using APA.

State transition pattern notation for APA

For the definition of the state transition relation of an elementary automaton e Є E, one has to specify all states of components C Є (e) (state components belonging to e) where e is active, i.e. can perform a state transition, and the changes of the states caused by the state transition. APA transition pattern notation is formally defined in (Gürgens et al., 2002a).

A specification of a state transition pattern consists of the *name of the transition pattern*, a *role identifier*, some *predicates* for the conditions to be checked and some *expressions* to describe the changes in the neighbour state components.

A state transition can occur when all expressions are evaluable and all conditions are satisfied. All possible variants of bindings of variables to elements of the state components are generated automatically, so if for example a component contains different hosts and a variable is used to represent a chosen source host and another variable is used for an arbitrary target host of an exploit then all possible combinations of source and target host are computed and further evaluated.

APA state transition patterns specify attacker and system behaviour

This paper is primarily concerned with using state transition patterns to model attacker behaviour but as a possible extension other types of state transition patterns are also considered that can be used to model the behaviour of enterprise network components. To reflect the different purposes of the state transitions three different types are distinguished here. They are characterised by the *role* that is associated with the transition type. An instance of a transition furthermore has a *name* to identify it; this can be for example the name of the exploit it specifies.

T Attacker *Exploit* specifications of atomic exploits based on the given vulnerabilities model the actions an attacker can take in arbitrary order; note that more than one attacker can act in that role

T Defence *Operation* specify a model for system defence strategy, tools and components (optional)

T Service *Action* a model of critical services the system provides (optional)

For a state transition pattern T Attacker *Exploit* modelling an exploit, a template structure was developed, so that additional exploits can easily be added following that layout. This template can serve as a basis to develop an automatic mechanism that generates such patterns from a knowledge base containing specifications of the known exploits.

In contrast to the generic nature of $\boxed{\text{T Attacker } \mathit{Exploit}}$, the state transition patterns $\boxed{\text{T Defence } \mathit{Operation}}$ and $\boxed{\text{T Service } \mathit{Action}}$ are individual for the modelled enterprise network, therefore no specific structure is assumed here. They reflect the state changes triggered by the respective operations.

Structure of state transition patterns for atomic exploits

Figure 4 shows a graphical representation of the template for $\boxed{\text{T Attacker } \mathit{Exploit}}$ including the neighbourhood relation (depicted by the edges) to the state components $\boxed{\text{S1}}$ - $\boxed{\text{S5}}$ (depicted by the circles) listed in the information model above.

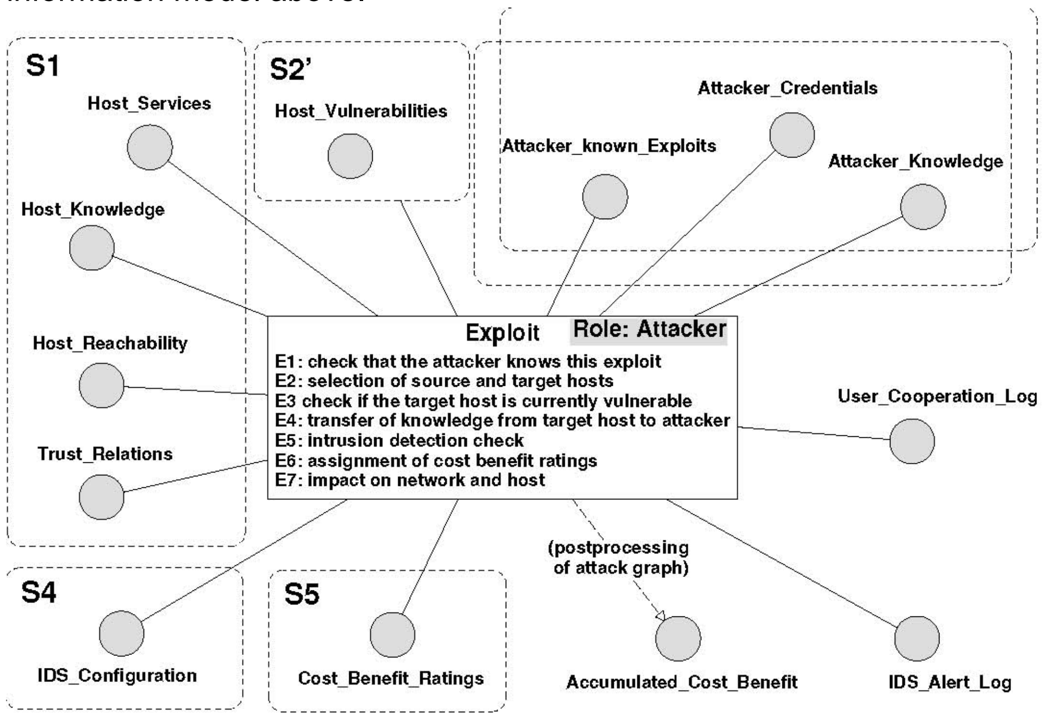

S1
Host_Services
Host_Knowledge
Host_Reachability
Trust_Relations
S2'
Host_Vulnerabilities
Attacker_Credentials
Attacker_known_Exploits
Attacker_Knowledge
Exploit
Role: Attacker
E1: check that the attacker knows this exploit
E2: selection of source and target hosts
E3 check if the target host is currently vulnerable
E4: transfer of knowledge from target host to attacker
E5: intrusion detection check
E6: assignment of cost benefit ratings
E7: impact on network and host
User_Cooperation_Log
(postprocessing of attack graph)
S4
IDS_Configuration
S5
Cost_Benefit_Ratings
Accumulated_Cost_Benefit
IDS_Alert_Log

Figure 4. Transition pattern template for exploit modelling.

According to this template, a state transition pattern modelling an exploit is constructed from the *role identifier*, here *Attacker* the *name* of the transition pattern which is identical to the *name of the exploit* and a body that comprises the following expressions:

$\boxed{\text{E 1}}$ a check that the attacker *knows* this exploit;
this is determined by an initial configuration that can be given directly or computed from a given set of exploits

$\boxed{\text{E 2}}$ a selection of source and target hosts for the exploit

- the source host is chosen from the host set the attacker already has adequate access to (in some cases the target also needs access to the source host for example to read a Trojan web page)
- the target host is chosen so that if the exploit succeeds the attacker will win some credentials or additional knowledge
  ➔ induces monotone growing attacker knowledge (no cycles in attack graph), therefore reduces complexity (see also (Ammann et al., 2002))

E 3 a check if the target host is vulnerable as stated in the specification of the vulnerabilities needed by this exploit (possibly multiple different exploits can be based on the same vulnerability)

E 4 the transfer of knowledge from target host to attacker; it has to be decided how to cope with changing knowledge of the captured host; is knowledge transferred once the host is captured or is a link from attacker to host knowledge inserted, so that the attacker always gets the updated contents ? Is attacker knowledge ever invalidated or is knowledge only valid for a time interval ? These questions influence the attack graph and may lead to cycles.

E 5 an intrusion detection check for that exploit

E 6 an assignment of cost benefit ratings to this exploit

E 7 an expression to implement the additional *impact on the network and host*; for example, to shut down or manipulate a host based intrusion detection system

The vulnerabilities checked in step E 3 above are represented in form of specifications representing the CVE/CAN vulnerabilities. These specifications include preconditions about the target host as well as network preconditions and describe effects that the vulnerabilities have on the attacker and possibly on the network and target host. A vulnerability is described by expressions with the following structure:

V 1 a check if the *target host is configured vulnerable*

- the target host has installed a product or products that are vulnerable with respect to the given vulnerability
- if necessary other preconditions are checked; for example, it could be essential for a vulnerability that a trust relation is established (as for example used in remote shell hosts allow/deny concepts)

V 2 a check if the target *host is currently running the respective products* (for example a vulnerable operating system or server version); if a user interaction is required this includes a check if the vulnerable product is currently used (for example a vulnerable internet explorer)

V3 a check for necessary *network preconditions*, including a check if the target host is reachable on the port the vulnerable product is using from the host the attacker selected as source
➔ this implicitly includes firewall rules (the model could be extended to explicitly model firewalls through extra transitions but this would blow up the state space significantly)

V4 an expression to cover the *effects for the attacker* ; for example, to obtain additional user or root credentials on the target host

V5 an expression to implement the *direct impact on the network and host*; for example, to shut down a service caused by buffer overflow

Attacker knowledge and behaviour

Attacker capabilities are modelled by the knowledge of exploits and hosts and the credentials on the known hosts that constitute the attackers profile. Knowledge of hosts changes during the computation of the attack graph because the attacker might gain new knowledge when capturing hosts. For example, if the attacker captures a portal or a host used as a firewall or a gateway he gets all information this host has. On the other hand, some knowledge may become outdated because the enterprise system changes ipnumbers or other configuration of hosts and reachability. 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. Optionally it is possible to specify extra transitions modelling an assumed impact an attacker might produce as for example shut down intrusion detection systems, send wrong or misleading information, shut down other services, denial of service attacks or other actions. But all this blows up the computation space and should be carefully used.

Monotonicity and invalid knowledge

It is not clear what is the best strategy to cope with dynamically changing configuration of hosts. To try keep the attacker knowledge monotone growing and get an attack graph without loops it is useful to model the knowledge as applicable only for some time interval but then if for example a host could change its ip-address arbitrarily the attack graph always grows with each change.

Assembling components of the model

The applied specification method based on *asynchronous product automata (APA)* is supported by the *SH verification tool* developed at the Fraunhofer Institute for Secure Telecooperation” (Ochsenschläger et al., 1999, 2000a). This tool provides components for the complete cycle from formal specification to exhaustive validation. The tool has been adapted and extended for the presented field of application.

The project management of the SH verification tool allows to select alternative parts of the specification and automatically “glues” together selected parts of the specified components (see figure 1) to generate a combined model of enterprise network specification, vulnerability and exploit specification and attacker specification. This can be used to answer $\boxed{Q2}$, $|Q3|$ and $|Q4|$ (see introduction). A very flexible selection of variants of analysis scenarios is implemented. The components are listed in a project tree and can be (de)activated by mouseclick. So it is easy for example to exchange libraries of specified vulnerabilities and exploits to analyse different versions and combinations of formal models and even compare different computed attack graphs or abstractions thereof in the analysis component of the tool.

Computation of attack graphs

After an initial configuration is selected, the attack graph (reachability graph) is automatically computed by the SH verification tool according to the definition in figure 2.

Two extra transitions that have turned out to be very useful have been included in the model as preprocessing steps. One computes the vulnerabilities per host from the information on products installed per host and vulnerabilities per product, the other generates 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.

To stop computation automatically when specified conditions are reached (or invariants are broken), so called break conditions can be specified using regular expressions. A violation of a security property for example, can in many cases be specified as a break condition.

For a quick check if something went wrong with the definition of the model, some statistic information is collected during computation of the graph. It can be used to find out, what state transitions appeared how often and what different values have been assigned to the state components during the computation.

Analysis of an attack graph

The main purpose of attack graph analysis is, to provide support for the persons in charge to assess the risks and the effects of possible countermeasures for the threatened network infrastructure.
The methodology for the analysis of an attack graph presented here that is outlined in figure 5 supports that process. It assists in revealing complex attack combinations and supports the systematic evaluation of possible solutions to minimise risk with given resources.

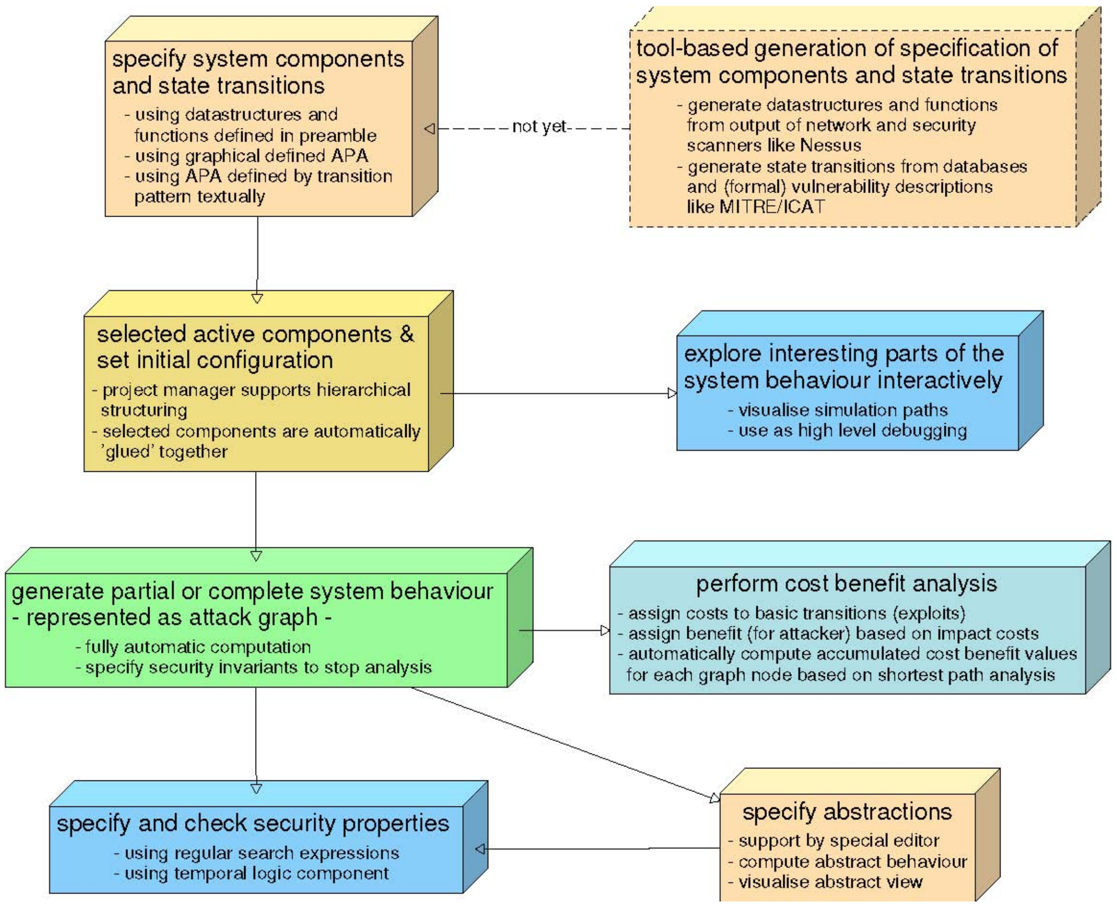

specify system components and state transitions
- using datastructures and functions defined in preamble
- using graphical defined APA
- using APA defined by transition pattern textually
not yet
tool-based generation of specification of system components and state transitions
- generate datastructures and functions from output of network and security scanners like Nessus
- generate state transitions from databases and (formal) vulnerability descriptions like MITRE/ICAT
selected active components & set initial configuration
- project manager supports hierarchical structuring
- selected components are automatically 'glued' together
explore interesting parts of the system behaviour interactively
- visualise simulation paths
- use as high level debugging
generate partial or complete system behaviour - represented as attack graph -
- fully automatic computation
- specify security invariants to stop analysis
perform cost benefit analysis
- assign costs to basic transitions (exploits)
- assign benefit (for attacker) based on impact costs
- automatically compute accumulated cost benefit values for each graph node based on shortest path analysis
specify and check security properties
- using regular search expressions
- using temporal logic component
specify abstractions
- support by special editor
- compute abstract behaviour
- visualise abstract view

Figure 5. Computation and analysis of attack graphs.

In the following paragraphs it is shown how to find answers to the questions posed in the introduction through analyses that can be accomplished after an attack graph is successfully computed. Many other interesting evaluations can be performed without question.

Finding violations of security properties

Security is not a singular property of a system. Depending on precisely what capabilities an attacker has, different properties for the system model have to be proven.

Formal specification of properties

System properties that are explicitly given by breakconditions can be checked during computation of the attack graph. Alternatively, security properties given in form of search queries, B ¨uchiautomata or temporal logic formulae can be verified after the graph is computed.

Finding states violating a safety (security) property

If a security property can be specified by a regular expression so that it is possible to check for a violation by inspecting a single node or edge then the property can be proven by a simple “search query” on the reachability graph. Often this can be supported in the model by collecting necessary information during the computation of the graph.

Model checking

If it is required to inspect some or all paths of the graph to check for the violation of a security property, as it is usually the case for liveness properties, then the temporal logic component of the SH verification tool can be used. Temporal logic formulae can also be checked on the abstract behaviour (under a simple homomorphism). A method for checking approximate satisfaction of properties fits exactly to the built-in simple homomorphism check (Ochsenschläger et al., 1999).

These methods provide appropriate support to answer question $\boxed{\text{Q 1}}$ from the introduction and are also helpful to research into many other questions.

Abstraction and visualisation of attack graphs

Abstraction capabilities of the SH verification tool support the definition of mappings, summarising or omitting transitions in the attack graph. The result is a view focused on some interesting aspect of the behaviour of the system. Technically this is implemented as a computation of the minimal automaton for an abstraction of the reachability graph that is specified via alphabetic language homomorphisms (Ochsenschläger et al., 2000b).

It is possible for example, to map multiple exploits with the same effects onto the same subsuming activity like “get-root-access”. This can be used to answer questions like $\boxed{\text{Q 7}}$ from the introduction. Another example is, to omit all exploits that are not detected by some intrusion detection component, in order to get a graph showing only the traces that an attack correlation component would see. Abstractions can also be defined using predicates. It is possible for example, to omit all transitions below a certain costbenefit ratio using an appropriate predicate.

Analysing IDS pattern detection

The transition patterns representing atomic exploits are modelled to include the behaviour of intrusion detection components, therefore their behaviour and their coaction to recognise attack pattern can be analysed. This helps to answer the question $\boxed{\text{Q 5}}$ (What attacks are detected ?) from the introduction.

Detections that are directly related to an atomic exploit are visible in the attack graph, because an intrusion detection check E 5 is included in each transition modelling an atomic exploit.

In more complex cases, evidence of attacks against the network is scattered over several atomic exploits on one host or several different hosts. The installed intrusion detection systems therefore have to collect and correlate information from different sources (Krügel and Toth, 2002).

Analysing the attack graph with regard to the required security properties leads to a detection of the paths that violate those properties. Abstraction helps to filter out the information concerning intrusion detection and gives a graph that visualises the correlation that is required to detect these violations. Now a scheme of coaction of intrusion detection components to detect this malicious behaviour or a superordinated component that checks for combined patterns can be designed.

Question Q 6 (What are the effects of changes to intrusion detection systems on overall detection of attacks ?) can be answered by comparing intrusion detection analysis of different attack graphs computed for different configurations selected in the project management component. It is useful to combine several features supported by the SH verification tool to answer this question. To filter out the intrusion detection information, abstractions of the different attack graphs are required. Based on this abstracted behaviour, a comparison of the behaviour of different versions is possible. The tool supports a comparison of those graphs and additionally the results of search queries and model checking helps finding the effects in question, but this task requires careful modelling, abstraction and finding the right properties to check.

Simulation

If the attacker has too many alternatives or the network is too complex, the state space of the composition of the selected specifications and their complex interplay may become too big to compute the complete behaviour. In this case it is appropriate to inspect selected parts of the state space. Simulation of interesting attack combinations is possible by interactive selection of paths in the visual representation of the part of the attack graph already computed and automatic proceeding in the selected direction. Other variants of simulation are also supported by the tool (for instance rando*m driven*). The seamless transition between verification and simulation on the same model is a particular strength of the approach presented here.

Cost benefit analysis

Cost benefit analysis as described in this paragraph is meant as a means to help 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. 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, the shortest (least expensive) path from be computed and visualised. This helps to answer question $\boxed{\text{Q 8}}$ (What is the attack with the *least costs* breaking a given security property ?) from the introduction.

A benefit for the attacker based on the negative impact he achieves can also be assigned, for example to indicate the *worth* regarding relative importance of the captured host.

Summarised costs and benefits can be compared for selected paths or the whole graph. For example searching for the node with the greatest benefit for the attacker answers question $\boxed{\text{Q 9}}$ from the introduction.

Comparing some configurations with available intrusion detection systems placed at different locations and computing attack graphs only for undetected attacks can help to decide what is a better position for the intrusion detection systems when looking at the maximum benefit for the attacker being undetected in the different scenarios (see also $\boxed{\text{Q 10}}$ from the introduction). To find a good covergage of intrusion detection given restricted resources, only relative evaluation of some predefined variants is intended here. It is shown in (Jha et al., 2002) that to decide which minimal set of security measures would guarantee the safety of the system is polynomially equivalent to the minimum hitting set problem (NPcomplete).

Survivability analysis

So far it was assumed that the enterprise network system does not react during an attack. This is in general a useful assumption to keep the graph of the system behaviour manageable. However the following extensions to the model can give valuable insight into related problems.

Game: System against attacker

In some cases it is interesting to consider some counterplay of the system. In Germany for example an ipaddress for a dslconnection is allocated dynamically and automatically changed every 24 hours. If for example the hosts of some teleworkers are part of the modelled enterprise system it is useful to check what effect this behaviour has on the attack analysis. Also if an attack is time consuming, it is possible, that it will be detected not only by an intrusion detection system but also possibly by some other security scanner tool or a human administrator checking the given configuration at certain time intervals. It is desirable to augment the model by some counteraction to describe for example a cut of a network connection in critical cases or the reconfiguration of a system.

Mission critical eservices

It is often very important, that even when an enterprise network system is under attack, at least some mission critical eservices survive that attack. Therefore it is essential, that it is possible to augment the attack scenario to include actions of the critical eservice and to analyse the extended scenario.

To verify if a given eservice survives an attack, a formal model of its components and their interplay must be added to the system model. The combined model can then be analysed by computing its dynamic behaviour and examining the generated state space. New safety and usually also liveness properties that constitute the required behaviour of the eservice have to be specified and verified. This helps in answering for example question $\boxed{\text{Q 11}}$ from the introduction. A methodology for developing an e-service so that it is robust against attacks has been described in (Rieke, 2003).

Because of the well known state space explosion problem, the extended scenarios have to be specified on a high abstraction level in order be able to compute the complete reachability graph. To find an appropriate abstraction level, it is essential to incorporate the hints given in the previous section concerning representative hosts. One should also consider to summarise similar attacks onto a representative abstract attack. For example only use the abstract attacks “get-user-access”, “get-root-access” and “from-user-to-root-access”.

Specification and analysis of an example scenario

To illustrate the methods described so far, a small example scenario is given now. The components are specified, the respective attack graph is described and some typical analysis outcome is sketched.

Scenario specification

Figure 6 shows the example scenario with the enterprise hosts named ms_host, nix_host, portal, db_server located inside the enterprise network and the host telework connected from the internet as well as the attacker. Vulnerabilities of the hosts needed for specification part $\boxed{S\ 2'}$ derived from the products installed and the product vulnerabilities are denoted below the host-names.

The installed intrusion detection components for specification part $\boxed{S\ 4}$ are depicted in figure 6 by the rhombic nodes. IDS_type1 is a network based system that detects exploits named CAN_2003_0693_ssh_exploit and rsh_login attempts. One IDS of that type is installed between the internet and the host portal, the other is installed to control the traffic between the portal and the host db_server. Furthermore a host based intrusion detection component IDS_type2 that detects exploits of type CAN_2002_0649_sql_exploit is installed directly on host db_server.

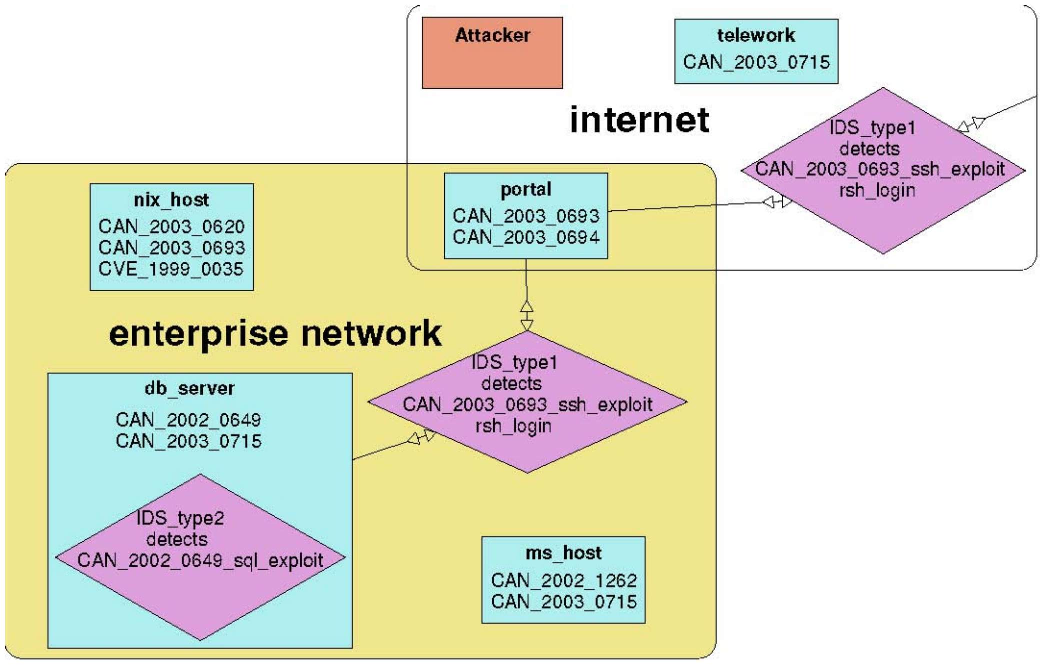


Figure 6. Example scenario

More information for specification part $\boxed{S\ 1}$ is provided by the tables in figure 7, showing the reachability of ports on all hosts and the active services. Some abbreviations are used here, namely zone_internet is an abbreviation for the hosts telework, attacker, portal and zone_intern is used for portal, nix_host, db_server and ms_host. The abbreviation port_all means reachability for all ports and finally the abbreviation net means physically connected.

Knowledge to be captured is only available on the portal that knows the addresses of all hosts. This could be used for example by the attacker to find out the dynamic allocated address of the telework host, that might be not so well administrated as the enterprise hosts directly connected to the network.

Host	Service	Port	User
telework	netbios ssnd	netbios ssn port	root
nix host	ftpd	ftp port	root
nix host	sshd	ssh port	root
nix host	rshd	rsh port	root
db server	ftpd	ftp port	root
db server	rshd	rsh port	root
db server	sql res	ms sql m port	db user
ms host	dcom		root
ms host	netbios ssnd	netbios ssn port	root
portal	sendmaild	smtp port	root
portal	sshd	ssh port	root

Source Host	Target Host	Port
zone internet	zone internet	port all
zone all	portal	ssh port
zone all	portal	smtp port
portal	zone intern	port all
zone intern	zone all	net
zone intern	zone intern	ftp port
zone intern	zone intern	rsh port
zone intern	zone intern	ssh port
db server	ms host	rpc port

Figure 7. Host reachability and installed services.

Attacker profile

It is assumed that the attacker knows all exploits that are specified in detail be low, namely CAN_2002_0649_sql_exploit, CAN_2003_0620_man_db_exploit, CAN_2003_0693_ssh_exploit, CAN_2003_0693_ssh_exploit_stealth, CAN_2003_0694_sendmail_exploit, CAN_2003_0715_dcom_exploit, CVE_1999_0035_ftp_exploit and the pseudo exploit rsh_login.

In the initial configuration the attacker has root credentials on the host attacker and no other access. The attacker nows the static addresses of all hosts except the dynamic address of the host telework. The attacker has no other knowledge.

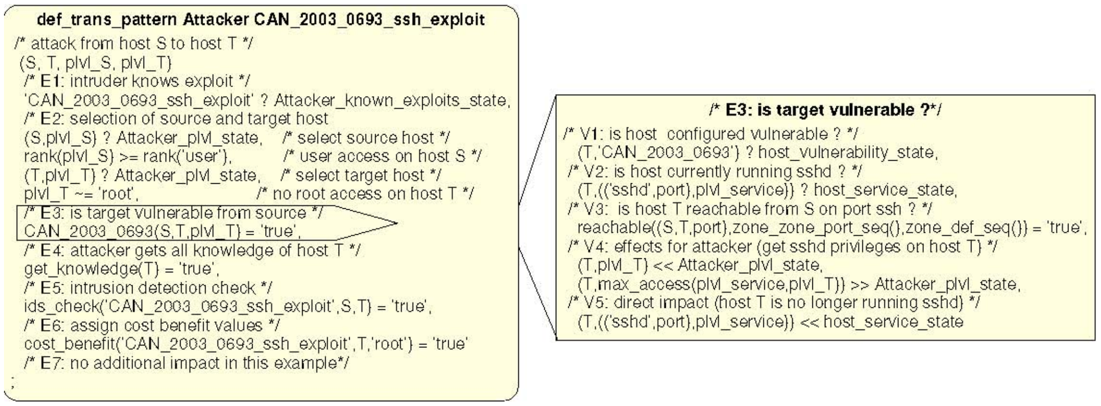
This completes the specification part $\boxed{S\ 3}$.

Vulnerabilities and exploits

The vulnerabilities and exploits described below are used in the example sce-nario. They are not described in detail here; more details are found at *MITRE* (see http://cve.mitre.org/) and *ICAT* (see http://icat.nist.gov/) sites.

Vulnerability CVE_1999_0035, an error in ftpd allowing to read/write arbitrary files is used to manipulate files to establish remote shell trust and this in turn used in combination with the rsh_login which is not a real vulnerability but a weak configuration to get remote access. This old vulnerability has been included because this example was used in some of the papers cited in the section on related work, to make it easier to compare different approaches. The related exploit using this vulnerability is named CVE_1999_0035_ftp_exploit.

The vulnerabilities CAN_2003_0620 (a buffer overflows in mandb) and the related exploit CAN_2003_0620_man_db_exploit, CAN_2003_0693 (a buffer management error in OpenSSH) and the related exploits CAN_2003_0693_ssh_exploit and CAN_2003_0693_ssh_exploit_stealth, CAN_2003_0715 (a heap-based buffer overflow in DCOM) and the related exploit CAN_2003_0715_dcom_exploit, CAN_2003_0694 (a buffer overflow in sendmail) and the related exploit CAN_2003_0694_sendmail_exploit as well as CAN_2002_0649 (buffer overflows in SQL server) and the related exploit CAN_2002_0649_sql_exploit are used to directly get access rights on a remote host. An example of the implementation of an exploit in SH verification tool syntax is given in figure 8.



```
def_trans_pattern Attacker CAN_2003_0693_ssh_exploit
/* attack from host S to host T */
 (S, T, plvl_S, plvl_T)
  /* E1: intruder knows exploit */
  'CAN_2003_0693_ssh_exploit' ? Attacker_known_exploits_state,
  /* E2: selection of source and target host
  (S,plvl_S) ? Attacker_plvl_state,    /* select source host */
  rank(plvl_S) >= rank('user'),        /* user access on host S */
  (T,plvl_T) ? Attacker_plvl_state,    /* select target host */
  plvl_T ~= 'root',                    /* no root access on host T */
  /* E3: is target vulnerable from source */
  CAN_2003_0693(S,T,plvl_T) = 'true',
  /* E4: attacker gets all knowledge of host T */
  get_knowledge(T) = 'true',
  /* E5: intrusion detection check */
  ids_check('CAN_2003_0693_ssh_exploit',S,T) = 'true',
  /* E6: assign cost benefit values */
  cost_benefit('CAN_2003_0693_ssh_exploit',T,'root') = 'true'
  /* E7: no additional impact in this example*/
;
```

```
/* E3: is target vulnerable ?*/
/* V1: is host  configured vulnerable ? */
  (T,'CAN_2003_0693') ? host_vulnerability_state,
 /* V2: is host currently running sshd ? */
  (T,(('sshd',port),plvl_service)) ? host_service_state,
 /* V3:  is host T reachable from S on port ssh ? */
  reachable((S,T,port),zone_zone_port_seq(),zone_def_seq()) = 'true',
 /* V4: effects for attacker (get sshd privileges on host T) */
  (T,plvl_T) << Attacker_plvl_state,
  (T,max_access(plvl_service,plvl_T)) >> Attacker_plvl_state,
 /* V5: direct impact (host T is no longer running sshd) */
  (T,(('sshd',port),plvl_service)) << host_service_state
```

Figure 8. Transition pattern for CAN_2003_0693_ssh_exploit.

Analysis of the scenario

Attack graph of the example scenario

The computed attack graph for this scenario has 142 nodes and 544 edges. Figure 9 shows a small section of it. The oval nodes depict single states, the rectangular nodes depict states with a hidden subgraph that can be expanded by mouseclick. The red (dotted) nodes mark states where the attacker has been detected by an intrusion detection component.

Check security properties

As an example for a security property to be checked for the scenario it is assumed that it is essential that an attacker can not gain any access at the db_server. The search query

```
({Attacker_plvl_state:<y>|  sfind((’db_server’,’db_user’),y) =0},,
 {Attacker_plvl_state:<x>|  sfind((’db_server’,’db_user’),x) >0});
```

checks if there are transitions in the graph where the attacker gains access as db_user at the db_server. For this query 66 matching edges are found.

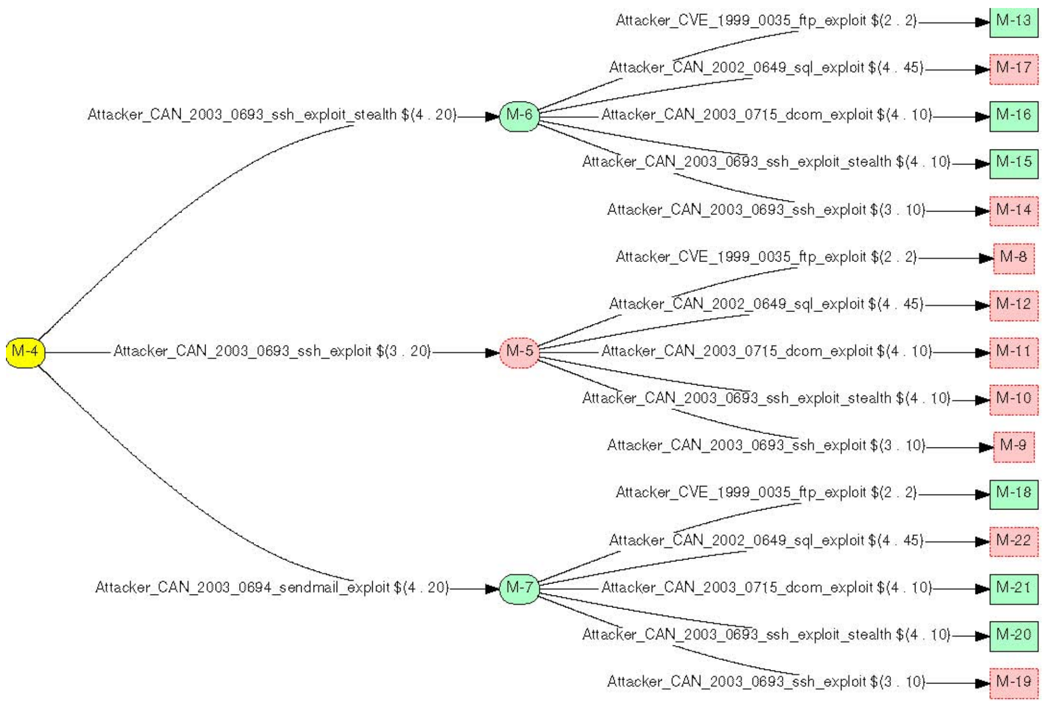

Attacker_CVE_1999_0035_ftp_exploit $(2 . 2)
M-13
Attacker_CAN_2002_0649_sql_exploit $(4 . 45)
M-17
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 20)
M-6
Attacker_CAN_2003_0715_dcom_exploit $(4 . 10)
M-16
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 10)
M-15
Attacker_CAN_2003_0693_ssh_exploit $(3 . 10)
M-14
Attacker_CVE_1999_0035_ftp_exploit $(2 . 2)
M-8
Attacker_CAN_2002_0649_sql_exploit $(4 . 45)
M-12
M-4
Attacker_CAN_2003_0693_ssh_exploit $(3 . 20)
M-5
Attacker_CAN_2003_0715_dcom_exploit $(4 . 10)
M-11
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 10)
M-10
Attacker_CAN_2003_0693_ssh_exploit $(3 . 10)
M-9
Attacker_CVE_1999_0035_ftp_exploit $(2 . 2)
M-18
Attacker_CAN_2002_0649_sql_exploit $(4 . 45)
M-22
Attacker_CAN_2003_0694_sendmail_exploit $(4 . 20)
M-7
Attacker_CAN_2003_0715_dcom_exploit $(4 . 10)
M-21
Attacker_CAN_2003_0693_ssh_exploit_stealth $(4 . 10)
M-20
Attacker_CAN_2003_0693_ssh_exploit $(3 . 10)
M-19

Figure 9. Attack graph of example scenario (small section).

Maximal impact

The computed attack graph for the example scenario has 18 so called *dead markings*. In cases where the graph has no loops these are the leafs of the graph. They denote states where no further processing occurs because the attacker has no more applicable atomic exploits available or has already captured all hosts.

Selecting an arbitrary dead marking and let the tool generate a way to the root node produces a path as shown in figure 10. The edge labels denote the atomic exploit chosen in that step as well as the target and source host. The first 2 edges represent state transitions for preprocessing steps as explained in the section on computation of attack graphs. The red (dotted) nodes M85, M122, M140 denote states where the attacker has already been detected. There is much more information available for each transition but this is hidden here by a presentation abstraction to keep the example readable. The numbers after the $-sign are explained in the next paragraph.

Inspecting the attackers knowledge at the dead marking M140 shows that he gained root access on hosts attacker, ms_host, nix_host and portal and furthermore he gained db_user access on db_server but none on telework.

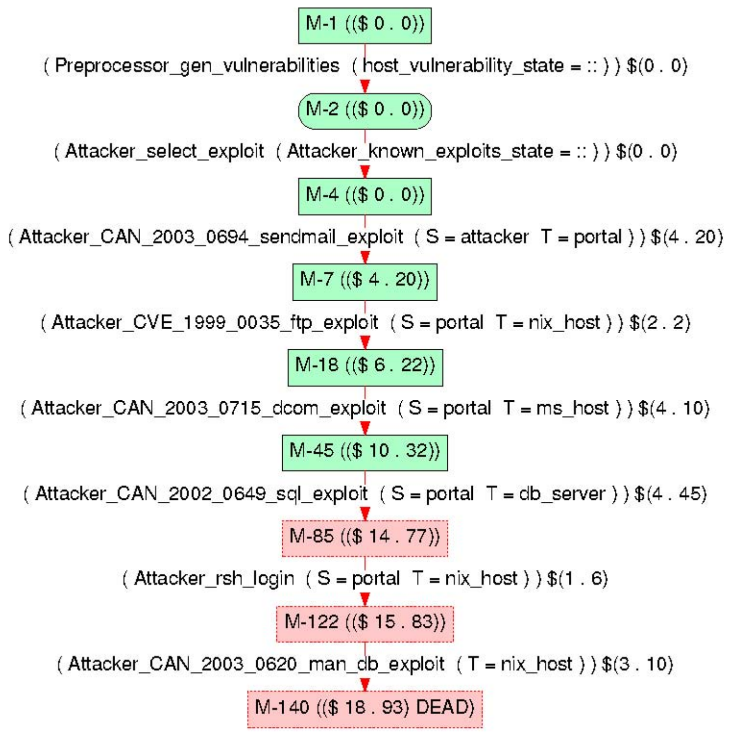

M-1 (($ 0 . 0))
(Preprocessor_gen_vulnerabilities (host_vulnerability_state = ::)) $(0 . 0)
M-2 (($ 0 . 0))
(Attacker_select_exploit (Attacker_known_exploits_state = ::)) $(0 . 0)
M-4 (($ 0 . 0))
(Attacker_CAN_2003_0694_sendmail_exploit (S = attacker T = portal)) $(4 . 20)
M-7 (($ 4 . 20))
(Attacker_CVE_1999_0035_ftp_exploit (S = portal T = nix_host)) $(2 . 2)
M-18 (($ 6 . 22))
(Attacker_CAN_2003_0715_dcom_exploit (S = portal T = ms_host)) $(4 . 10)
M-45 (($ 10 . 32))
(Attacker_CAN_2002_0649_sql_exploit (S = portal T = db_server)) $(4 . 45)
M-85 (($ 14 . 77))
(Attacker_rsh_login (S = portal T = nix_host)) $(1 . 6)
M-122 (($ 15 . 83))
(Attacker_CAN_2003_0620_man_db_exploit (T = nix_host)) $(3 . 10)
M-140 (($ 18 . 93) DEAD)

Figure 10. Path to root with cost benefit notations.

Cost benefit evaluation

For cost benefit evaluations an adequate measure has to be defined. In the example scenario it is assumed, that costs reflect the effort an attacker uses in each step of the attack. Costs are directly assigned to the atomic exploits in this example, whereas the benefit for a transition is computed as the worth of the target host multiplied by the rank of the access right gained. The benefit for the attacker reflects the negative impact for the enterprise. Of course other kinds of measures for cost and benefit or other appropriate measures could be implemented following the proposed scheme. Assumed costs and benefits per exploit for specification part $S5$ of the example scenario are assigned as shown in the tables in figure 11.

Exploit	Cost
CAN_2003_0693_ssh_exploit	3
CAN_2003_0693_ssh_exploit_stealth	4
CVE_1999_0035_ftp_exploit	2
CAN_2003_0620_man_db_exploit	3
CAN_2003_0715_dcom_exploit	4
CAN_2003_0694_sendmail_exploit	4
CAN_2002_0649_sql_exploit	4
rsh_login	1

Host	Worth
telework	1
attacker	0
nix_host	2
ms_host	2
db_server	9
portal	4

Access	Rank
none	1
restricted_user	2
user	3
db_user	4
root	5

Figure 11. Cost benefit values.

Now by shortest path computation in a postprocessing step on the attack graph, the values for cost and benefit can be summed up along the paths with the least cost to any node. The cost and benefit of a transition is depicted by the numbers after the $sign at the edges in figure 10 that shows an example path in the attack graph with annotated cost benefit annotations. The sums along the path are depicted inside the nodes in the same figure.

A search for the node with the highest benefit score for the attacker (where most negative impact is achieved) returns the node M135 which has the same benefit rating namely 93 as the node M140 in figure 10.

Cut down the graph

Defining a condition that stops further computation after an attack has been detected by an intrusion detection component generates only a subgraph with 57 nodes (33 dead) and 95 edges. The graph reduced to only undetected attacks generates a subgraph with only 24 nodes (4 dead) and 60 edges.

Cost benefit analysis for the graph with undetected attacks shows that the maximum benefit an attacker can obtain undetected in this scenario is 48. Inspecting the respective node in the attack graph shows that the attacker has gained root access on hosts attacker, ms_host, nix_host and portal but no access on db_server and telework.

Abstraction

In some applications the SH verification tool already computed graphs of about 1 million edges in acceptable time and space. But it is impossible to visualise a graph of that size. So abstraction focussing on some interesting aspect is definitely a comfortable way to go in this case. An example for the usage of behaviour abstraction is shown in figure 12. The abstract view in this case shows that only one type of exploit can be used to attack the db_server and the graph is reduced from 544 edges to only one edge in the abstracted behaviour. The predicate used to define the corresponding mapping hides (maps to epsilon) all transitions that don't have the target host db_server.

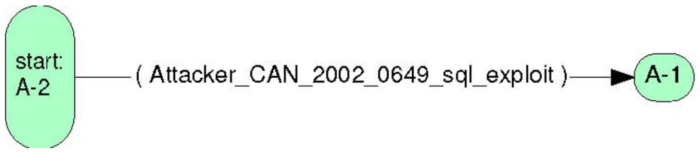


Figure 12. Attack graph abstraction showing transitions with target db_server.

Figure13 shows an abstraction focussing on transitions with benefit > 10 and the resulting graph is also very concise.

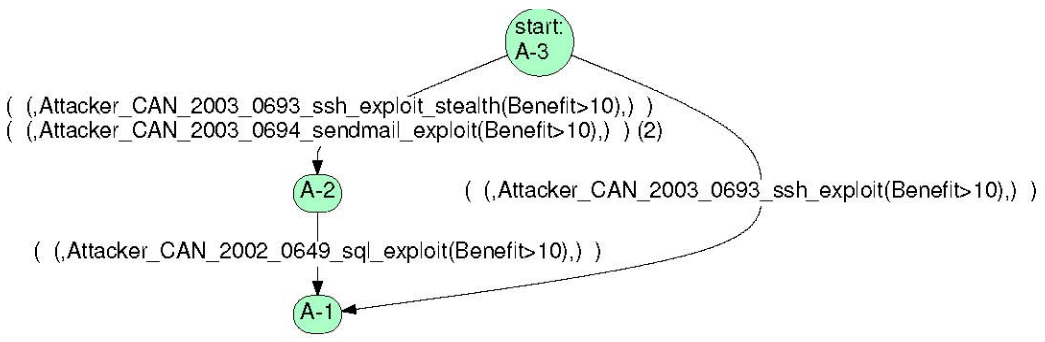

start: A-3
((,Attacker_CAN_2003_0693_ssh_exploit_stealth(Benefit>10),))
((,Attacker_CAN_2003_0694_sendmail_exploit(Benefit>10),)) (2)
A-2
((,Attacker_CAN_2003_0693_ssh_exploit(Benefit>10),))
((,Attacker_CAN_2002_0649_sql_exploit(Benefit>10),))
A-1

Figure 13. Attack graph abstraction showing transitions with benefit >10.

System countermeasures and critical services

As an example for a check for critical services availability and to demonstrate how system countermeasures can be added to the framework defined so far, it is assumed that the host db_server always tries to answer queries from host teleworker. As a precondition the server checks if sshd is running on the portal because a "ssh-tunnel" on that host is used to reach teleworker. Now as shown in figure 8 (in condition V 5) the attacker kills the sshd when executing the CAN_2003_0693_ssh_exploit. So if the attacker applies this exploit to attack the host portal, then afterwards the sshd is not active on that host and so db_server cannot send an answer to telework anymore. Now additionally a system countermeasure is considered that restarts the sshd on the portal from time to time. Two transitions patterns, namely T Defence *Restart_sshd* and T Service *Answer* add these actions to the model. The defence operation restarts sshd when it is down and the service action checks for an active sshd on portal. No other details are added to keep the model small.

Now a new computation of the attack graph results in a graph with 234 nodes (0 dead !) and 1136 edges. A section of this graph is shown in figure 14.

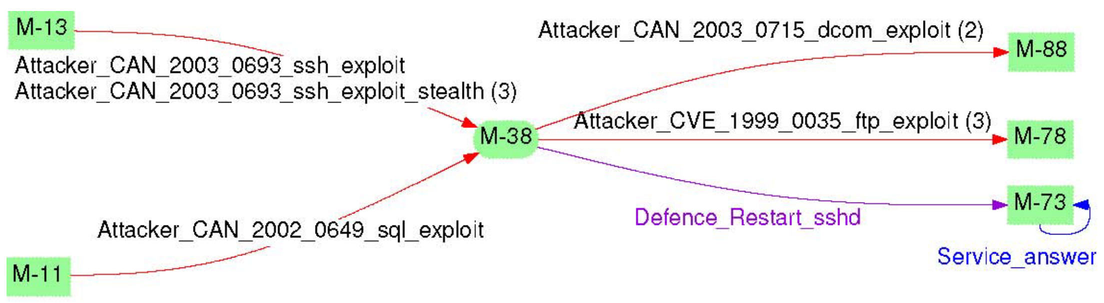

M-13
Attacker_CAN_2003_0693_ssh_exploit
Attacker_CAN_2003_0693_ssh_exploit_stealth (3)
M-38
Attacker_CAN_2003_0715_dcom_exploit (2)
M-88
Attacker_CVE_1999_0035_ftp_exploit (3)
M-78
Defence_Restart_sshd
M-73
Service_answer
Attacker_CAN_2002_0649_sql_exploit
M-11

Figure 14. Section of attack graph with service and countermeasure.

A typical liveness question for the sketched situation is Q 11 from the introduction (Is a *client* still able to get answers from a DB-server when the enterprise network is under attack ?). Using an appropriate type of model checking, *approximate satisfaction* of temporal logic formulae can be checked by the SH verification tool (Ochsenschläger et al., 1999, 2000a). In terms of temporal logic the property above can be written as G F Service_Answer (always eventually Service_Answer) which is found to be true by the tool.

Lifting the assertion that the attacker only attacks a host if he gains some credentials for the CAN_2003_0693_ssh_exploit (see figure 8 the check for “no root access” on target host in E 1 E 2) leads to an attack graph with 3062 nodes (0 dead) and 22228 edges. This illustrates the dramatic influence of monotonicity assumptions on attack graph growth.

Related work

The approach that Phillips and Swiler first presented in (Phillips and Swiler, 1998) is closest to the approach proposed in this paper. They described a prototype tool implementing their method in (Swiler et al., 2001). Similar to the computation method based on the SH verification tool outlined here, their method computes an attack graph starting from an initial node, but they don’t describe abstraction methods to visualise compact presentations of the graph and they don’t address liveness analysis that is used here to assure system response to critical services under attack.

Jha, Sheyner, Wing et al. use scenario graphs in (Jha and Wing, 2001) and attack graphs (Jha et al., 2002; Sheyner et al., 2002) that are computed and analysed based on model checking.

Ammann et al. presented an approach in (Ammann et al., 2002) that is focussed on reductions of complexity of the analysis problem from exponential to polynomial by explicit assumptions of monotonicity.

Conclusions

Within the critical infrastructure protection context this paper aims at the protection of the core information infrastructure although the methods presented here could be extended and applied to other types of infrastructure and threats. The presented methodology for computation and analysis of attack graphs outlined in figure 5 is based on a formal specification of an organisation's critical network infrastructure, supplemented by a generic vulnerability and exploit specification and an attacker specification to model the threats against that infrastructure. The tool supported analysis of the attack graph assists in revealing vulnerabilities of the threatened infrastructure including complex attack combinations and supports the systematic evaluation of possible solutions to minimise risk with given resources. Contributions of this work are:

Specification framework for critical network infrastructures and threats

It is worked out in detail, how to formally specify topology and components of the information infrastructure and represent it by state components in asynchronous product automata (APA) notation. The operational formal system specification is completed by specifications of vulnerabilities, exploits and attacker capabilities represented by APA state transitions (see figures 1, 3 and 4). Specific templates to support and simplify formal modelling of enterprise networks under attack have been developed. Moreover, extensions to the model to add system defence operations and critical services actions are proposed, supplemented by some abstraction concepts, to prevent state space explosion problems in such models.

Methodology and tool to analyse vulnerabilities and countermeasures

From the APA specification an attack graph representing the behaviour of the model is automatically computed. Based on this graph, tool supported analysis methods are presented that can be used to answer the various questions posed in the introduction. Specific features of this approach comprise:

- an integrated interactive visualisation support to browse or debug the behaviour of the model and explore selected parts of the graph
- the usage of a well-elaborated and formally proven abstraction concept combined with an appropriate model checking component for analysis of security and liveness properties
- an integrated costbenefit analysis method
- a seamless transition between verification and simulation on the same model when a complete computation of the attack graph is not possible
- a flexible configuration management simplifies evaluation and comparison of different solutions

Further research objectives

To seamlessly integrate the methods and tool presented here into a network vulnerability analysis framework, a toolassisted transformation of a system configuration as provided by administration databases or gathered by network scanners into formal specifications is required. Likewise, some improvement towards generic formal vulnerability and exploit specifications is needed.

An in-depth research objective is, to develop methods and tool support to reduce state space explosion by further elaborating the ideas on abstraction of the system specification as sketched in the paragraphs about "representative hosts". For such a *tool assisted specification abstraction*, it has to be (automatically) proven, that the system specification is *appropriately transformed* into the abstracted specification, to assure that system properties are transported from a lower to a higher level of abstraction and no critical behaviour is hidden.

Another interesting perspective is, to extend the specification and analysis method described in this paper for application in other similar structured sce-narios, as for example, to model a networked infrastructure system of a country including specifications of mutual dependencies as described in (Luiijf et al., 2003).Such a model could be used to analyse vulnerabilities and to raise risk awareness. It could help to reveal complex attack combinations and support systematic evaluation of possible solutions. This approach aims at optimising security and protection of networked systems with given resources.

References

Paul Ammann, Duminda Wijesekera, and Saket Kaushik (2002). Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 217–224. ACM Press New York, NY, USA. ISBN 1581136129.

Urs E. Gattiker, Hervé Debar, Gasper Lvarencic, Giannis A Pikrammenos, Jerman Borka, Roland Rieke, Atta Badii, Yong Hua Song, Theis Søndergaard, Rainer Fahs, Helga Treiber, and Mario Wolframm (2003). Cyber attack methods detection & information exploitation research project proposal. URL http://www.eicar.org/camdier/index.html.

S. Gürgens, P. Ochsenschläger, and C. Rudolph (2002 a). Authenticity and Provability - a Formal Framework. GMD Report 150, Fraunhofer-Institute for Secure Telecooperation.

S. Gürgens, P. Ochsenschläger, and C. Rudolph (2002 b). Role based specification and security analysis of cryptographic protocols using asyn chronous product automata. In *DEXA 2002 International Workshop on Trust and Privacy in Digital Business*. DEXA. URL http://www.sit.fhg.de/english/META/meta_publications/doc/Dexa2002abstract.pdf Copyright: © 2002, IEEE. All rights reserved.

Somesh Jha, Oleg Sheyner, and Jeannette M. Wing (2002). Two formal analyses of attack graphs. In *15th IEEE Computer Security Foundations Workshop (CSFW15 2002), 2426 June 2002, Cape Breton, Nova Scotia, Canada*, pages 49–63. IEEE Computer Society.

Somesh Jha and Jeannette M. Wing (2001). Survivability analysis of networked systems. In *Proceedings of the 23rd international conference on Software engineering*, pages 307–317. IEEE Computer Society.

Christopher Krügel and Thomas Toth (2002). Distributed pattern detection for intrusion detection. In *Network and Distributed System Security Symposium Conference Proceedings: 2002*, 1775 Wiehle Ave., Suite 102, Reston, Virginia 20190, U.S.A.. Internet Society. URL citeseer.nj.nec.com/501183. html

E. Luiijf, H. Burger, and M. Klaver (2003). Critical infrastructure protection in the netherlands: A quickscan. In *EICAR Conference Best Paper Proceedings*.

Benjamin Morin, Ludovic Mé, Hervé Debar, and Mireille Ducassé (2002). M2d2: A formal data model for ids alert correlation. In *Recent Advances in Intrusion Detection, 5th International Symposium, RAID 2002, Zurich, Switzerland, October 1618, 2002, Proceedings*, volume 2516 of *Lecture Notes in Computer Science*, pages 115–137. Springer.

P. Ochsenschläger, J. Repp, and R. Rieke (2000a). The SHVerification Tool. In *Proc. 13th International FLorida Artificial Intelligence Research Society Conference (FLAIRS2000)*, pages 18–22, Orlando, FL, USA. AAAI Press. ISBN 01577351134.

Peter Ochsenschläger, Jürgen Repp, and Roland Rieke (2000b). Abstraction and composition – a verification method for cooperating systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 12:447–459.

Peter Ochsenschläger, Jürgen Repp, Roland Rieke, and Ulrich Nitsche (1999). The SHVerification Tool AbstractionBased Verification of Cooperating Systems. *Formal Aspects of Computing, The International Journal of Formal Method*, 11:1–24.

Cynthia A. Phillips and Laura Painton Swiler (1998). A graph-based system for network-vulnerability analysis. In *NSPW '98, Proceedings of the 1998 Workshop on New Security Paradigms, September 2225, 1998, Charlottsville, VA, USA*, pages 71–79. ACM Press.

Roland Rieke (2003). Development of formal models for secure eservices. In *Eicar Conference 2003*. URL http://www.sit.fhg.de/english/META/ meta_publications/doc/Eicar2003.pdf.

Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jean-nette M. Wing (2002). Automated generation and analysis of attack graphs. In *2002 IEEE Symposium on Security and Privacy, May 1215, 2002, Berkeley, California, USA*, pages 273–284. IEEE Comp. Soc. Press.

Laura P. Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian (2001). Computerattack graph generation tool. In *DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume 2,June 12 14, 2001, Anaheim, California*, pages 1307–1321. IEEE Computer Society.