

# Collaborative Attack Modeling<sup>\*</sup>

Jan Steffan  
IT Transfer Office  
Department of Computer Science  
Darmstadt University of Technology  
steffan@ito.tu-darmstadt.de

Markus Schumacher  
IT Transfer Office  
Department of Computer Science  
Darmstadt University of Technology  
schumacher@ito.tu-darmstadt.de

## ABSTRACT

Avoidance and discovery of security vulnerabilities in information systems requires awareness of typical risks and a good understanding of vulnerabilities and their exploitations. In this paper we compare common methods of sharing security related knowledge with regard to their ability to support avoidance and discovery of vulnerabilities. We suggest a new method of collaborative attack modeling that is especially suitable for this purpose. This method combines a graph-based attack modeling technique with ideas of a Web-based collaboration tool.

## Keywords

security, attack modeling, collaborative knowledge management

## 1. INTRODUCTION

Frequent reports about security vulnerabilities show that still many deficits exist in the development of secure software systems. A recent example is the huge number of format-string and buffer-overflow vulnerabilities. The problem is even more pressing as the attacker activity and the destructiveness of attacks, such as Distributed Denial of Service, have increased over the last years.

A central problem in the design of secure software systems and the security evaluation of existing systems is the danger of overlooking weak links in the security chain. Using multiple cascaded safeguards, or redundant chains to stay with the chain-metaphor, is one approach to lower the risk of undiscovered vulnerabilities. The other is to make the chain as strong as possible by preventing weak links. In this paper our focus is on the second approach.

Discovering vulnerabilities in a complex software system or in a network is much more complicated than in a physical

---

<sup>\*</sup>This project was supported by T-Systems/Deutsche Telekom AG

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC 2002, Madrid, Spain  
© 2002 ACM 1-58113-445-2/02/03... \$5.00

chain. That is where the analogy ends. One thing is that it is not easy to discover all parts of a system that are relevant for its security, or in other words whole chain links may be overlooked. In practice it is very common that a system's security is compromised through a path its developers never have thought of.

Another difference to the chain-metaphor is that two system parts may be without any flaw from a security point of view individually, but become fatal for the system's security in combination. Configuration errors are a typical cause for this kind of situation. Discovering potential security vulnerabilities is a process that requires creativity and good knowledge of the system, its parts and their interdependencies. Unfortunately, this knowledge is often spread over many people such as developers, implementors, administrators, or support staff, which typically do not belong to the group of persons doing security evaluations.

Finally, there is no simple way of measuring the strength of a system part's security. The main ingredient for a thorough security evaluation is expert knowledge on typical security weaknesses and their exploits. The more experts bring in their experience, the more likely it is that security flaws are discovered. The generally high security quality of open source software is an example of this principle.

In order to avoid vulnerabilities in the first place, developers, administrators etc. have to be aware of the causes of vulnerabilities and possible exploits.

In short, there are three measures that can help discovering and avoiding security vulnerabilities:

- having many experts contribute and share their knowledge,
- making expert knowledge about typical vulnerabilities available to developers and
- making system-specific knowledge available to persons searching for vulnerabilities.

We are presenting an approach that extends a WikiWeb-like system for collaborative work on informal documents with a graph-based attack modeling method. This way we provide a common place where both security and development people can organize their knowledge in a consistent way.

This paper is divided into four main parts. In the following section we will discuss commonly used methods for sharing security-related knowledge. In section 3 we do a more detailed analysis of graph-based attack modeling techniques, which is the most suitable approach for our purpose. In section 4 our own approach to Web-based security knowledge sharing is presented. A summary and possible directions of future work are given in section 6.

## 2. RELATED WORK

There are various methods in use for sharing security-related knowledge. As summarized in table 1, they differ in their goals, the covered aspects and their suitability for certain application areas. We will now give an overview of commonly used knowledge sharing techniques and analyze them with regard to their benefit to the prevention of security vulnerabilities.

	attack models	programming guidelines	pattern languages	evaluation criteria	vulnerability databases
vuln. avoidance:	⊕⊕	⊕	⊕	⊖	⊖
vuln. discovery:	⊕⊕	⊕	⊖⊖	⊕	⊖
sec. evaluation:	⊕	⊖	⊖⊖	⊕⊕	⊕⊕
mistakes:	⊕⊕	⊕⊕	⊕	⊖⊖	⊕
measures:	⊖⊖	⊕	⊕⊕	⊕⊕	⊖⊖
general:	⊕⊕	⊕	⊕⊕	⊕⊕	⊖⊖
system specific:	⊕⊕	⊖	⊖	⊖⊖	⊕⊕
design-level:	⊕	⊖	⊕⊕	⊕⊕	⊖⊖
code-level:	⊕⊕	⊕⊕	⊖	⊖⊖	⊖⊖

**Table 1: Strengths of common security knowledge representation methods**

### *Graph-based attack models*

Graph-based representations capturing the steps of a successful attack range from simple tree models [8, 4] to more formal petri net based methods [6, 5].

What distinguishes attack modeling approaches from other methods is that the viewpoint of an attacker is taken. As the attacker’s goals and methods play the central role in nearly all security considerations [7], attack models are closest to the problem and therefore are useful for the discovery of vulnerabilities in new systems, the avoidance of vulnerabilities during software development, and for the evaluation of installations for known vulnerabilities.

Attack models can capture both general and system specific attack methods as well as the system properties and other preconditions that make a successful attack possible.

Due to their compact graphical representations, attack models often lack explanations and background knowledge that can be better communicated with written documents such as programming guidelines and pattern languages. Attack models also focus on the causes of vulnerabilities, but do not provide solutions for counter-measures. Attack models therefore should be complemented by programming guidelines or security patterns and informal problem descriptions.

### *Programming guidelines, check-lists, security pattern languages*

Security patterns [10] and programming guidelines (such as [11]) capture expert knowledge in order to make it available to non-experts.

Patterns are problem-oriented and therefore – in contrast to guidelines – primarily address questions above the source code level, such as system architecture, algorithms, protocols, security policy enforcement, or network structure. The abstraction levels of pattern languages and programming guidelines complement each other.

Evaluation criteria such as [2] usually provide checklists of general security measures. As the presence of security measures does not imply the absence of vulnerabilities, evaluation criteria are of limited value for the avoidance and discovery of vulnerabilities.

### *Vulnerability Databases*

Vulnerability databases such as Bugtraq [1] focus on the description of known vulnerabilities in certain systems, but usually lack detailed information about the how and why of a vulnerability and its exploitation. This makes vulnerability databases less useful for the avoidance and discovery of vulnerabilities in new systems [9].

A benefit of vulnerability databases over guidelines, pattern languages, and other forms of written documents is the low turnaround time between the discovery of a new vulnerability and its publication.

As attack models focus on the causes of vulnerabilities, they are most suitable for our goal of supporting discovery and avoidance of security vulnerabilities. Programming guidelines and pattern languages are less formal and can contain more detailed textual explanations. The advantage of vulnerability databases is their low turnaround time. Our approach of a Web-based attack modeling system combines all of those benefits.

## 3. GRAPH-BASED ATTACK MODELING

We will illustrate the differences between different graph-based attack modeling methods with the following simple attack scenario: An attacker has found a way to read the `/etc/passwd` file on a UNIX system *host*. This file contains the password hashes (no shadow passwords are used) and user *Alice* has chosen a weak password. This enables the attacker to brute-force guess the password of *Alice* and login on *host* with *Alice*’s privileges. This informal attack description contains a lot of information:

### *Steps of an attack and their order in time*

the first step is to get `/etc/passwd`, then to successfully guess a password and finally to log in on the system.

### *Interdependencies of attack steps*

successful brute-force password analysis is possible if the content of `/etc/passwd` is known *and* it contains password hashes *and* a user has chosen a weak password.

### *Preconditions of an attack*

the attacker’s ability to read `/etc/passwd` and the existence of a weak password.

### Postconditions of the attack

When the attacker logs in, he gains new privileges and capabilities (e. g. for installing a back-door).

### Attacker actions

some action has to be performed by the attacker to get from the knowledge of `/etc/passwd` to the knowledge of a password.

### Capabilities and resources required

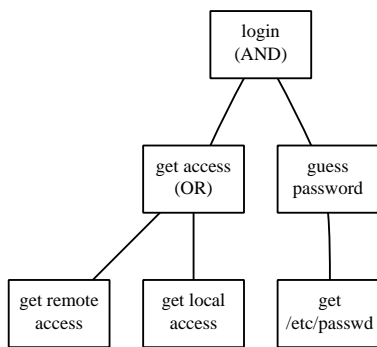
in order to do a successful brute-force password guessing the attacker requires sufficient computing power or a good dictionary.

## 3.1 Attack and Fault Trees

Fault trees have been used for the analysis of failure conditions of complex technical systems for a long time. An example of modeling attacks with a fault tree-like method is presented by Guy Helmer et al in [4]. Bruce Schneier was first to associate the term “Attack Tree” with the use of fault trees for attack modeling which made this approach more widely known [8].

Attack trees can capture the steps of an attack and their interdependencies. Attack trees are also used to represent and calculate probabilities, risks, cost, or other weightings.

The main building blocks of attack trees are nodes. In both [4] and [8] nodes are used to model steps of an attack or attacker actions. Each tree has a single top node which represents the achievement of the attack’s ultimate goal. In our example the top node would be starting a login shell on the target host (see figure 1). Interdependencies of goals are modeled by the tree hierarchy. Attack steps that have to be performed successfully before another step can occur are represented by child nodes. To each node either a logical AND or a logical OR gate is associated. An OR-node can occur when any of its child events occurs. For an AND-node to occur all of its child events are necessary.



**Figure 1: Attack tree representation of sample attack scenario**

Tree nodes can be augmented with probabilities or costs, so that the most likely or inexpensive attack path can be calculated. However, those weightings are too specific to be applied to attack trees describing general attack scenarios. In addition, in a collaborative environment it would be impossible to find a consensus on attack probabilities.

The attack tree approach has some advantages that make it a good choice for the representation of collaboratively managed attack knowledge. The hierarchical structure can simplify navigation and enable multiple experts to work on different branches of a tree in parallel. The concept of attack trees is easy to understand, allowing a wider range of persons to take advantage of an attack tree or to contribute to it.

We had a group of students collaboratively translate different attack scenarios into attack trees. Though the overall result proved the usefulness of attack trees, we identified the following problems:

One thing we missed was a way of modeling preconditions such as system properties that are required for an attack. In [8] possible/impossible flags are suggested to indicate whether some attack step (typically represented by a leaf node) can occur. Those flags indirectly reflect the properties of a given system, but no clue is given, why a node evaluates as possible or impossible.

The fact that attack actions and resulting states are both recorded in a single node easily leads to fuzzy node labels. In the above example the node `login` expresses that an attacker both gains some capabilities and has to perform some action to do so. However neither the capabilities nor the action are defined precisely.

Finally, the structure of AND/OR nodes is not suitable for frequent editing and extensions. Often, there are good reasons for a node to be both AND and OR node, for example when there are two alternative ways of achieving the node’s goal (OR), one of which is the conjunction of two subgoals (AND). To resolve this situation, the introduction of an artificial middle node for connecting the AND branch is required. It is often difficult to associate a goal or name with those artificial nodes. Moreover, adding an AND branch to an existing OR node can require restructuring existing child nodes, which is very inconvenient and confusing, making this form of attack trees unsuitable for collaborative and iterative editing.

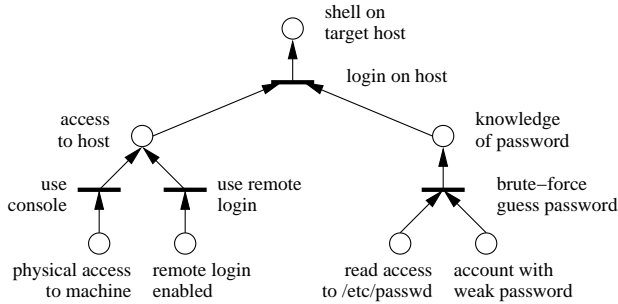
## 3.2 Petri Net Based Models

A petri net consists of places, transitions, arcs and tokens. Places are represented by circles and can be compared to nodes in a graph. Action or flow is modeled with tokens that move between places along transitions. Transitions are represented by rectangles and connected to places by arcs. Transitions have AND semantics, meaning that a token is passed only if a token is available at each of the input arcs.

The “Attack Net” model presented by McDermott in [6] is a straightforward approach to attack modeling with petri nets. In this model, attack steps are represented by places similar to nodes in attack trees. Transitions are used for the explicit modeling of attacker actions which significantly extends the expressiveness of this model compared to attack trees.

Petri nets are not restricted to tree structures, allowing the reuse of subgraphs in different contexts. In some cases even cyclic petri nets make sense for attack modeling. One exam-

ple is an attack in which the attacker uses the same strategy to jump from one machine to another inside a network.



**Figure 2: Attack net representation of password example**

Transitions can express all logical dependencies that can be modeled with distinct AND/OR nodes in attack trees, but they do not share their inflexibility. Figure 2 shows the attack net representation of our example attack scenario. Both AND- and OR-conditions can be added without changing the existing structure. For example, in order to include social engineering as an alternative way of getting knowledge of a valid password, a new transition can be added to the node *knowledge of password*. AND-conditions can be added by attaching new places to a transition, such as a new node hashes in */etc/passwd* to the transition *brute-force guess password*.

In [5] Helmer et al show how colored petri nets (petri nets with colored tokens) can be used for very comprehensive attack modeling. In this approach tokens are associated with tuples (“colors”). The tuples contain placeholders for the objects of an attack, such as target and source host. Transitions are augmented with complex logical conditions that make use of these placeholders.

It isn’t difficult to sketch a similar attack model based on predicate logic. States can be represented by predicates such as *can-login(Attacker, System)* while transitions are expressed as rules such as *has-weak-password(User, Password) ⇒ can-guess-password(Attacker, User)*. However those two simple expressions also demonstrate the drawbacks of this model in a collaborative environment: the predicates *can-guess-password* and *can-login* are clearly related, but some additional relation between *User* and *System* is needed to manifest this relation in the model. Inconsistent predicates, rules and variables can occur easily when multiple persons are contributing to a common knowledge base.

## 4. ATIKI – A TOOL FOR COLLABORATIVE ATTACK MODELING

ATiki is our new approach to a Web-based system that supports collection and sharing of security-related knowledge. ATiki combines an extended form of the attack net modeling method with a flexible Web-based collaboration platform based on the WikiWeb paradigm.

### 4.1 The WikiWeb

The WikiWeb [3] was developed as a discussion and collaboration platform for authors and users of design pat-

terns. The intention of the WikiWeb is the same as our’s: enabling many experts and non-experts to contribute and discuss ideas improves the quality and completeness of the overall result.

The WikiWeb approach is both simple and powerful. A WikiWeb is a set of cross-referenced, mostly text-based Web-pages. The difference to ordinary Web-pages is that each page can be edited at the spot directly in the browser by anyone who has permission to do so. The usual policy of WikiWebs is to allow anyone the editing of pages. This way not only a selected group of experts can contribute, but also occasional readers.

Pages are written in a special, easy to learn and to read markup language that allows simple text formatting such as headlines, numbered lists, or hyperlinks. Each page is identified by a unique “WikiName” which is a run-together capitalized title or phrase describing the page such as “Help-Page” or “BufferOverflow”. The system automatically renders WikiNames as hyperlinks.

The WikiWeb system implements some additional features that simplify navigation, like full text search, lists of references to a page, and a list of recent changes. The WikiWeb implementation we used also contains a revision system for pages that allows users to visualize changes made to a page and to view old revisions.

For our purpose, the WikiWeb system provides an ideal foundation. The WikiWeb allows its users to organize, update and improve the stored information in an interactive way. Information can be cross-referenced and commented in a way that would not be possible within a fixed database scheme. In addition, informal content and references to related or complementary external sources such as vulnerability databases or security guidelines can be included, which is a major advantage in our context.

### 4.2 The ATiki Attack Modeling Method

After practical investigation of the various attack modeling approaches described in section 2, we found the attack net method to be the one most suitable for collaborative attack modeling. Our main extensions to the attack net model are the explicit notion of pre- and postconditions and the introduction of a context notion that we will explain in the next section.

Like attack nets, the ATiki model has two major elements: *conditions* (corresponding to places in attack nets) and *transitions* which are both implemented as special Wiki pages. Arcs are substituted by hyperlinks between the corresponding Wiki pages.

Conditions describe system properties or attacker capabilities. Transitions are defined through pre- and postconditions and describe how a set of preconditions can lead to the given set of postconditions through an exploit or any other way. The semantics of transitions is the same as in attack nets, i. e., all of the preconditions have to occur in order to enable the transition to the postconditions.

Conditions are meant to express informal predicates on the

system (e.g. Unlimited failure logins are allowed) or an attacker's capabilities (valid password is known). A true/false label should be assignable to each predicate.

Associating Wiki pages with conditions and transitions has a number of advantages. Wiki pages can contain freely structured descriptions that can be augmented with detailed background information, code samples, discussion threads, or what else helps to increase the value and usefulness of the information. Another advantage is that each condition and transition can be identified and referenced by its WikiName. With this reference, conditions and transitions can be hyperlinked from within the ATiki attack net structure, from normal Wiki pages, and even from outside the WikiWeb.

In order to improve navigation and to enable users to get an overview of attack nets, we have supplemented the ATiki system with a graphical representation. Each condition or transition node in the graph is clickable and directs the user to the corresponding Wiki page. The graphical representation is generated on-the-fly so it always reflects the current structure of the attack net.

### 4.3 The Context Notion

The ATiki system is intended to cover information relevant to a wide range of environments as well as to specific software packages or versions. Therefore, conditions or transitions will make sense only in a more or less specific context, such as all flavors of UNIX systems, programs using some library, or a single installation. This leads to the following three requirements:

- A mechanism is needed to help users filter out relevant pieces of information.
- As conditions and transitions do not make sense in all situations, they have to be placed into context.
- The relationships between general vulnerabilities and occurrences in concrete systems have to be expressible.

In order to resolve these requirements we introduced a notion of context that is based on multiple inheritance similar to the class concept in object-oriented programming.

Like conditions and transitions, a context is represented by a special type of Wiki page. This page either holds a self-contained description of the context or references one or more base contexts. All properties of base contexts are inherited and specialized by adding a new property. In the example illustrated in figure 3 the context **RedHat Linux system** is a specialization of **Linux system** which itself inherits from **UNIX like system**. **RedHat v6.0** inherits from both **RedHat Linux system** and **Linux v2.2**.

Contexts are intended as a classification scheme for condition and transition pages as well as for ordinary Wiki pages. Pages can be augmented with one or more context references. Additional descriptions that are valid in the given context only, can be attached to the context reference as comments.

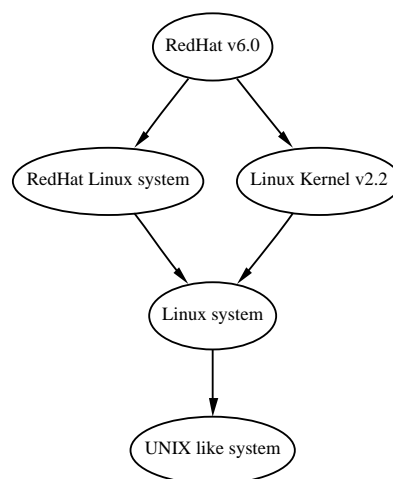


Figure 3: Example of context class hierarchy

The semantics of context inheritance is that pages associated with a context are also associated with specializations of this context that inherit from it. Pages that do not specify a context apply in any context.

In the sample UNIX context hierarchy from above this means that pages that are defined within the context **UNIX-like system** apply in the contexts **Linux system** and **RedHat Linux system**, too. This makes sense since an attack that is general enough to be applicable in any UNIX like environment is likely to work on a RedHat system, too.

Figure 4 shows a transition that could be part of the password example. The context **Linux system** is augmented with an additional comment applying to Linux systems only.

**Brute-force guess password**

**Preconditions:** [[→ read access to /etc/passwd](#)], [[→ account with weak password](#)]

**Postconditions:** [[→ knowledge of password](#)]

**Contexts:** [[→ UNIX-like system](#)], [[→ Linux system](#)]: most modern Linux systems use shadow passwords, so /etc/passwd does not contain password hashes.

**Description:** A password can be guessed if it is included in a reasonably small search space, such as all combinations of lowercase letters or lists of English words or names. See [[→ account with weak password](#)] for more cases of weak passwords.  
If the hash value of the password is known, an attacker can do the password guessing off-line by generating a hash value for each candidate in the search space and comparing it with the known hash.

Figure 4: Transition **Brute-force guess password** with pre- and postconditions and context. [[→ ...](#)] denotes hyperlinks in the ATiki system.

A user who is interested in information relevant to Linux systems can use the corresponding context as a starting point to get only those conditions and transitions that apply in this context. This will include all pages that specify Linux systems or UNIX like systems as their context, or that do not define any context at all. However, this user would not get pages specific to RedHat Linux. This filtering mode allows for pointed searches.

A disadvantage of filtering out contexts that are more specific than the current selection is, that in practice new security weaknesses are often discovered in a narrow context first and generalized later. It therefore is desirable to extend the horizon of selected pages towards specializations of a context.

In order to meet this requirement we introduced a threshold. The threshold defines how many levels of child classes are included in the scope of the currently selected context. The default is a threshold of zero that corresponds to a sharp selection as described in the above example. Setting the threshold to one would include RedHat Linux and any other context that directly inherits from Linux system, such as Linux Kernel v2.2. A threshold of  $n$  selects specializations of the current context upto  $n$  levels deep in the context inheritance graph.

Due to the class hierarchy the context of a page can be adjusted to a more general class later without affecting the original relationships.

The use of multiple inheritance avoids a common problem with the definition of hierarchical categories. There is often no clear choice between a number of possible specializations of a class. In the above example, the context Linux system is specialized both to RedHat Linux system and Linux v2.2. With multiple inheritance both specializations can be defined and reunified into a common context at a deeper level such as RedHat v6.0. As contexts in ATiki are defined and maintained by multiple persons with probably different viewpoints, this property of multiple inheritance is very important.

## 5. PRACTICAL EXPERIENCES

We have used a prototype of the ATiki system for the security evaluation of a PHP-based groupware system. As a first step we searched various resources such as Bugtraq and other security portals for PHP-specific vulnerabilities. We also investigated known vulnerabilities of other PHP-based systems and generalized them as far as possible. In this phase the ATiki system was useful in several ways:

- Many information sources left some open questions that required more research in other resources or a test setup. Undefined Wiki links could be inserted as placeholders for unclear issues and filled in later. The same was done for spontaneous ideas that required some more detailed investigation.
- Gaps that were left by one person could be filled in by someone else.
- While studying resources we came across pieces of in-

formation which were not relevant for our PHP investigation, but nevertheless worth keeping for later use. This information could quickly be added to the knowledge base at the right place without causing too much interruption of the primary work.

- By generalizing vulnerabilities as far as possible and breaking them down into elementary conditions and transitions we came to a precise understanding of the vulnerability and its causes.

After this initial research phase we analyzed the source-code of the PHP groupware system for occurrences of the determined potential vulnerabilities. In this phase we took advantage of the ATiki system in the following ways:

- Due to its mostly hierarchical structure, the attack net graph could be worked through similar to a list of evaluation criteria, which is helpful for organizing and documenting work.
- An advantage of the attack net structure compared to a list of criteria is that non-relevant subgraphs can be pruned early if some condition is not fulfilled. Another advantage is that related information can be found close to each other.
- As in the general research phase new insights could be included easily and immediately.
- A specific context was defined for the evaluated system. Each condition or transition that turned out to be valid for the system was marked with this context. This way it is possible to generate list or attack net representations of all issues relevant for this system. By defining appropriate sub-contexts the same can be done for submodules of the groupware system.
- Descriptions of the occurrences of conditions in the evaluated system (e.g. which function in which file is vulnerable) could be added in commented context references.

Finally the ATiki system was helpful in producing a report on the identified vulnerabilities:

- Parts of the attack net graph were quite useful for the presentation and visualization of vulnerabilities and their causes. However, attack net graphs can easily grow to a size that does not fit onto paper. Contexts can help to select manageable subgraphs.
- The text of the report could be based on the notes and descriptions collected in the previous phases.

In addition, there is the added value of a knowledge base on vulnerabilities in PHP-based systems. This can be used and extended in later PHP-related evaluation or development projects.

## 6. CONCLUSION AND FUTURE WORK

Most security vulnerabilities could be avoided or discovered early if knowledgesharing between and inside the groups of system engineers and security experts could be improved. We have investigated knowledgesharing techniques currently used in the area of security and showed that the attack net model is most suitable for this purpose.

We extended the attack net method and combined it with the WikiWeb collaboration tool in order to enable groups of persons to work on the same attack net model. The advantages of the attack net structure are that it allows pieces of information to be sorted in, combined, and looked up easily. The WikiWeb system complements the more formal attack net model with informal descriptions and the freedom to reference internal or external background information. The WikiWeb concept is also the basis for collaborative work and supports this with change logs, revision control, collision handling, full text search, and other features.

With the context concept a second information structuring method orthogonal to the attack net model was introduced. The context structure allows classification and filtering of information. With the inheritance mechanism, information at different levels of abstraction including system specific facts can be set into relation.

So far we have used the context technique for the definition of the system environment only. Nevertheless, it may be used for other aspects of classification, too. For example, a classification of attackers could be made based on their capabilities, resources, or motivation. By assigning attacker profiles to conditions and transitions, risk analyses similar to those suggested in [8] would be possible.

A huge number of public WikiWeb systems (see [3] for references) has shown that the idea of publicly editable content works very well in practice and does not suffer from vandalism as it might be expected. It will be interesting to see whether this applies to a public version of ATiki, too, which requires more discipline by its users in order to keep the content consistent and well structured. Measures like restricted write access, moderation, or a voting system might be useful in order to ensure high quality content.

## 7. REFERENCES

- [1] Bugtraq Vulnerability Database. <http://www.securityfocus.com>.
- [2] Common Criteria Project Homepage. <http://www.commoncriteria.org>.
- [3] CUNNINGHAM, W. The WikiWikiWeb. <http://c2.com/cgi-bin/wiki>.
- [4] HELMER, G., WONG, J., SLAGELL, M., HONAVAR, V., AND MILLER, L. A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System. In *Symposium on Requirements Engineering for Information Security* (Indianapolis, USA, 2001).
- [5] HELMER, G., WONG, J., SLAGELL, M., HONAVAR, V., MILLER, L., AND LUTZ, R. Software Fault Tree and Colored Petri Net Based Specification, Design and Implementation of Agent-Based Intrusion Detection Systems. <http://citeseer.nj.nec.com/helmer01software.html>.
- [6] MCDERMOTT, J. Attack Net Penetration Testing. In *The 2000 New Security Paradigms Workshop* (Ballycotton, County Cork, Ireland, Sept. 2000), ACM SIGSAC, ACM Press, pp. 15–22.
- [7] SALTER, C., SAYDJARI, O., SCHNEIER, B., AND WALLNER, J. Toward a Secure System Engineering Methodology. Technical report, Counterpane Systems, 101 East Minnehaha Parkway, Minneapolis, MN 55419, Sept. 1998. New Security Paradigms Workshop.
- [8] SCHNEIER, B. Attack Trees. *Dr. Dobbs's Journal of Software Tools* 24, 12 (Dec. 1999), 21–29.
- [9] SCHUMACHER, M., HAUL, C., HURLER, M., AND BUCHMANN, A. Data Mining in Vulnerability Databases. In *7. Workshop "Sicherheit in vernetzten Systemen"* (Hamburg, Germany, March 2000), DFN-CERT. <http://www.dvs1.informatik.tu-darmstadt.de/DVS1/research/sechouse/publications/sdb-dfn-cert-eng.pdf>.
- [10] SCHUMACHER, M., AND ROEDIG, U. Security Engineering with Patterns. In *8th Conference on Pattern Languages of Programs (PLoP 2001)* (Monticello, Illinois, USA, September 2001).
- [11] WHEELER, D. A. Secure Programming for Linux and Unix HOWTO. <http://www.linuxdoc.org/HOWTO/Secure-Programs-HOWTO/>, 2001.

## Appendix: Extensive Example

The following pages show part of the data collected in the ATiki system while evaluating the PHP-Groupware system. The last page shows the corresponding attack net graph for the context “PHP Groupware”.

### Contexts

*web-server*

**Specializes:** server-side application

**Description:** A server-side application that answers HTTP requests. PHP scripting language

*server-side web application*

**Specializes:** web-server

**Description:** A server-side application that enables a web-server to deliver dynamic content.

*PHP based application*

**Specializes:** server-side web application

**Description:** A server side web application that is based on the PHP scripting language

*MySQL based application*

**Specializes:** Application

**Description:** An application that is based on the MySQL database.

*PHP-Groupware*

**Specializes:** PHP based application, MySQL based application

**Description:** A web-based groupware system that is implemented in PHP and uses MySQL as database-backend. The homepage of the PHP Groupware project is at <http://phpgroupware.sourceforge.net>

### Conditions

*C1: Application has permission to open server side file*

**Contexts:** server-side application, server-side web application: Those typically have the privileges of the user running the web-server  
PHP Groupware: at least has permission to open its PHP scripts and configuration files.

**Description:** A server side application has permission to open a specific server side file.

*C2: Application opens Attacker defined server side file*

**Contexts:** server-side web application

**Description:** A server side application opens a server side file which is specified by an attacker, instead of the file intended by the developer.

*C3: Attacker can bypass input validation*

**Contexts:** server-side web application

**Description:** A server side application either doesn't verify user supplied input for validity at all or in an insufficient way that allows an attack to send potentially malicious input values.

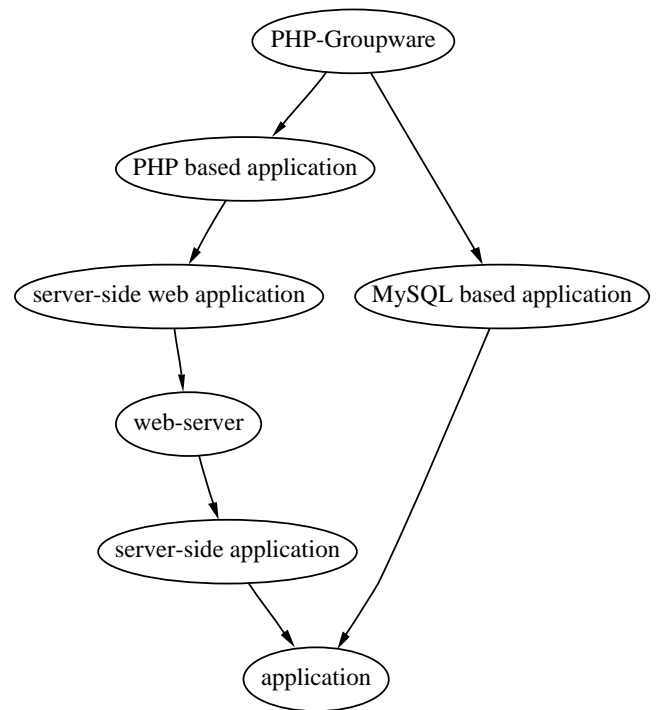


Figure 5: Context class hierarchy for PHP Groupware

*C4: Attacker can view server-side file*

**Contexts:** server-side web application

**Description:** An attacker can cause a server-side application to disclose a more-or-less arbitrary server-side file.

*C5: Web application allows file upload*

**Contexts:** server-side web application

PHP Groupware: file upload is permitted by the mail module (attach\_file.php) and the file-manager module.

**Description:** a server side web application allows remote web-site users to upload files over the HTTP protocol.

*C6: Application shows file*

**Contexts:** server-side web application

PHP Groupware: files can be shown through the mail and file-manager module.

**Description:** a server side application makes a server side file visible to remote web-site users.

*C7: Server-side file contains plain-text password*

**Contexts:** PHP Groupware: The database password is contained in the headers.inc config file

**Description:** a server-side file contains access credentials in plain-text, such as login name and password for a database.

*C8: Attacker can get password*

**Contexts:** PHP Groupware: The database password may become visible

**Description:** An attacker can get into possession of a valid password.

#### *C9: web-server discloses file*

**Contexts:** web-server

**Description:** Due to a configuration error a web-server delivers a server-side file that was intended not to be delivered or only after being processed in answer to an HTTP request.

#### *C10: processing of file fails*

**Contexts:** web-server, web application

**Description:** The processing of a server-side file by a web application fails.

#### *C11: file has undefined extension*

**Contexts:** web-server

**Description:** A server-side file has a filename pattern that is not associated with with an application (script interpreter, web-server module) that processes it. This can easily happen for forgotten backup-files with extensions like `.bak` or

### Transitions

#### *T1: Exploit PHP file upload vulnerability*

**Contexts:** PHP based application

**Preconditions:** C5 Web application allows file upload, C3 Attacker can bypass input validation

**Postconditions:** C2 Application opens attacker defined server-side file

**Description:** PHP stores uploaded files in a server side temporary file. The location and name of this file is stored in a well known (defined by file input field) POST or global variable. Manipulation of this variable by an attacker may be undiscovered by the script causing it to open a server-side file specified by the attacker. This problem was published as Bugtraq-ID 1649

**Example:** The HTML form:

```
<input type="file" name="upload">
```

when filled in with the local filename `foo` and submitted results in the PHP variables:

- `upload="/tmp/randomfile"` pointing to a file on the host that contains the uploaded data
- `upload_name="foo"` containing the original filename
- `upload_size` and `upload_type` containing the file size and mime-type

An attacker can make up an HTTP POST request (eg. by setting up a local html form) that binds `upload` to some other server-side filename:

```
<input type="hidden" name="upload" value="/etc/passwd">
```

**Counter-measure:** As a counter-measure PHP scripts accepting and processing file uploads should check that the server-side file is located in the expected (temp) directory and that the file size matches.

#### *T2: Cause application to disclose server-side file*

**Contexts:** server-side application

**Preconditions:** C1 Application has permission to open server-side file, C2 Application opens attacker defined server-side file, Application shows file

**Postconditions:** C4 Attacker can view server-side file

**Description:** —

#### *T3: View server-side file containing password*

**Contexts:** server-side web application

**Preconditions:** C7 Server-side file contains plain-text password, C4 Attacker can view server-side file

**Postconditions:** C8 Attacker can get password

**Description:** —

#### *T4: Get server-side file through HTTP request*

**Contexts:** web-server

**Preconditions:** C9 web-server discloses file

**Postconditions:** C4 Attacker can view server-side file

**Description:** Due to a configuration error an attacker can successfully request a server-side file that was intended to be delivered only after being processed (such as CGI- or PHP-scripts).

#### *T5: File is delivered unprocessed*

**Contexts:** web-server

**Preconditions:** C10 processing of file fails

**Postconditions:** C9 web-server discloses file

**Description:** —

#### *T6: Web-server doesn't know how to handle file*

**Contexts:** web-server

**Preconditions:** C11 file has undefined extension

**Postconditions:** C10 processing of file fails

**Description:** —

