

# SeaMonster: Providing tool support for security modeling

Per Håkon Meland  
SINTEF ICT, System development and security  
7465 Trondheim, Norway  
per.h.meland@sintef.no

Daniele Giuseppe Spampinato, Eilev Hagen, Egil Trygve Baadshaug,  
Kris-Mikael Krister and Ketil Sandanger Velle  
The Norwegian University of Science and Technology  
7491 Trondheim, Norway  
{spampina, eilev, egiltryg, krismika, ketilsan}@stud.ntnu.no

## Abstract

*Today there is a knowledge gap between security experts and software developers, one that is likely to widen as the number of security vulnerabilities and the complexity of software increases. Security models can be used to exchange knowledge between these two camps, however proper tool support is vital to achieve this. This paper presents an open source initiative to such a tool called SeaMonster, which uses well-known and easily understandable security modeling techniques. By providing and linking different viewpoints to vulnerabilities, different aspects can be expressed, such as causes, threats and countermeasures within the same tool. Our goal is to create a common platform for security modeling, increase the understanding of vulnerabilities and reduce the amount of time it takes to model security. As a consequence we hope to see a general improvement in the security of software.*

## 1 Introduction

The increasing use of information systems requires more software to be made now than ever before, but a side effect is that we are also introducing a large number of security vulnerabilities into the systems. Computers are to a great extent used to handle sensitive information and critical processes, making them increasingly dependent upon complex software and services. It is shown that with more complexity, the trend is larger exposure for vulnerabilities that can be exploited during intentional attacks [2]. According to The National Vulnerability Database [20], the amount of discovered vulnerabilities is over 30 times higher today compared to ten years ago, and the number is continuously rising. Even though there are

---

*This paper was presented at the NISK-2008 conference.*

many good practices for improving software security, the released software does not seem to have any noticeable improvements when it comes to security [15].

The traditional way of protecting the software is by relying on network security solutions, but merely trusting firewalls and anti-virus applications will not hold in the long run. During software development, security is usually something added near the end as an afterthought. Most researchers in the software security field agree with McGraw when he states [11]; *Because security is an emergent property of a complete system, we must take a holistic approach. We should weave in security throughout the complete software development lifecycle.* It is also a well-known fact that the estimated cost of late error correction is considerably higher than in earlier development phases [8].

There are few software developers who follow the secure coding best practices [7], even though most of the vulnerabilities are caused by well-known mistakes which we already have solutions for [5]. This is due to that software developers, who are experts in their field, are not necessarily experts in security, and vice versa [2]. Additionally, there is also a general lack of security experts in the software engineering field [15]. For these reasons, the typical software development projects do not necessarily have available security expertise to complement the ordinary developer's lack of knowledge (and interest) in the area.

We believe that security modeling is something that can help reduce this knowledge gap, since models are informative and reusable artifacts that can be used to identify, describe and plan how vulnerabilities can be avoided. Security modeling is especially beneficial during the early phases of software development, and can help determine the right level of security. Today, security modeling is often done using general purpose drawing tools [2]. This leads to different notations, no coupling between models and little tool support in order to avoid possible misinterpretations of the produced output.

This paper presents security modeling with a freely available tool called *SeaMonster* (a name creatively constructed from *security modeling software*). *SeaMonster*'s focus is on helping developers and security experts to easily model security both before and during any phase of software development, and to facilitate sharing and reuse of security models between projects, people and organizations. As a result to this we hope to see increased security awareness during the early phases of development, without having to add a lot of extra resources (which is often the case why security is down-prioritized in the first place).

The next section introduces *SeaMonster*, before we look at how it supports three different viewpoints of security modeling. This is followed by a discussion comparing it to other tools used to model security, and also mentions some planned improvements. The paper is then finalized by a conclusion.

## 2 **SeaMonster explained**

*SeaMonster* is a graphical modeling application that lets you describe and present different aspects related to security vulnerabilities. These aspects are called viewpoints. Viewpoints are used to create a view, which is what you see when looking from the chosen viewpoint. A viewpoint can have several views, represented by different models. In *SeaMonster* there are three different viewpoints that complement each other:

- What causes the vulnerability.
- How your system is threatened and can be attacked because of the vulnerability.

- How the vulnerability or the effects of the vulnerability can be mitigated through countermeasures.

Different modeling techniques can be used to express these views, and we would like to emphasize that we have not invented any new techniques. SeaMonster uses a selection that security experts already are familiar with, but adds the functionality of linking them together through the different viewpoints based on a common information model. SeaMonster is highly modularized, and additional notations and security modeling techniques can be added based on a plugin architecture.

SeaMonster is built on top of the Eclipse framework<sup>1</sup>, which is a collection of different frameworks and tools implemented as plugins. The two main frameworks are the Graphical Modeling Framework (GMF) and the Eclipse Modeling Framework (EMF). The SeaMonster application has been released under an open source license, and originates from student work at the Norwegian University of Science and Technology (NTNU) and the associated applied research foundation SINTEF. It is continuously being improved and the project is hosted by SourceForge<sup>2</sup>.

We will now explain the background of the three viewpoints, and show how they are expressed in SeaMonster. Our examples are based on the *cross-site scripting* (XSS) vulnerability, which can briefly be described as a way of injecting malicious code into a Web application. The exploit occurs when the code is executed by some user/victim, and the result can for example be that somebody takes over your online banking session. Grossman et al [6] state that *XSS arguably stands as the most potentially devastating vulnerability facing information security and business online*, and it is currently the dominating Web vulnerability.

### 3 Causes

This viewpoint is based on formalisms and methods in security modeling that focus on the causes of vulnerabilities. The goal of these formalisms is to get an in-depth understanding of what causes the vulnerabilities, so that they can be prevented. This is an activity typically performed by security experts after one or several similar vulnerability instances has been discovered.

Root Cause Analysis (RCA) is a collective term used to describe a wide range of approaches, tools, and techniques, which are used to uncover causes of a problem [1]. RCA is based on a belief that solving a problem is best done by correcting or eliminating its roots causes, as opposed to merely addressing the immediately obvious symptoms. It differs from troubleshooting and problem solving in that these disciplines typically seek solutions to specific problems, whereas RCA is directed at underlying issues.

Fault tree analysis (FTA) [9] is a method used in RCA. It is a logical, structured process that can help identify potential causes of a failure before the failure occurs. Faults, and causes to these, are visually modeled using the logic operators *AND* and *OR* to represent the sequence of faults and causes. Using this diagram it should be possible to backtrack one or more root causes to the event.

Vulnerability Cause Graphs (VCG) [4] are visual representations of vulnerability models, allowing developers and security experts to get an overview of the relations between vulnerabilities and their causes. In this representation, the causes and the vulnerabilities are presented in a directed acyclic graph with four kinds of nodes: *simple*, *compound*, *conjunction* and *exit* nodes. This view is supported by SeaMonster, and Figure 1 shows how we have modeled the XSS vulnerability. You can see that there are four potential causes to the XSS. In the leftmost case, a malicious Web site owner has deliberately placed

---

<sup>1</sup>Eclipse is an open development platform, see <http://www.eclipse.org/>

<sup>2</sup>The SeaMonster home page is <http://sourceforge.net/projects/seamonster/>

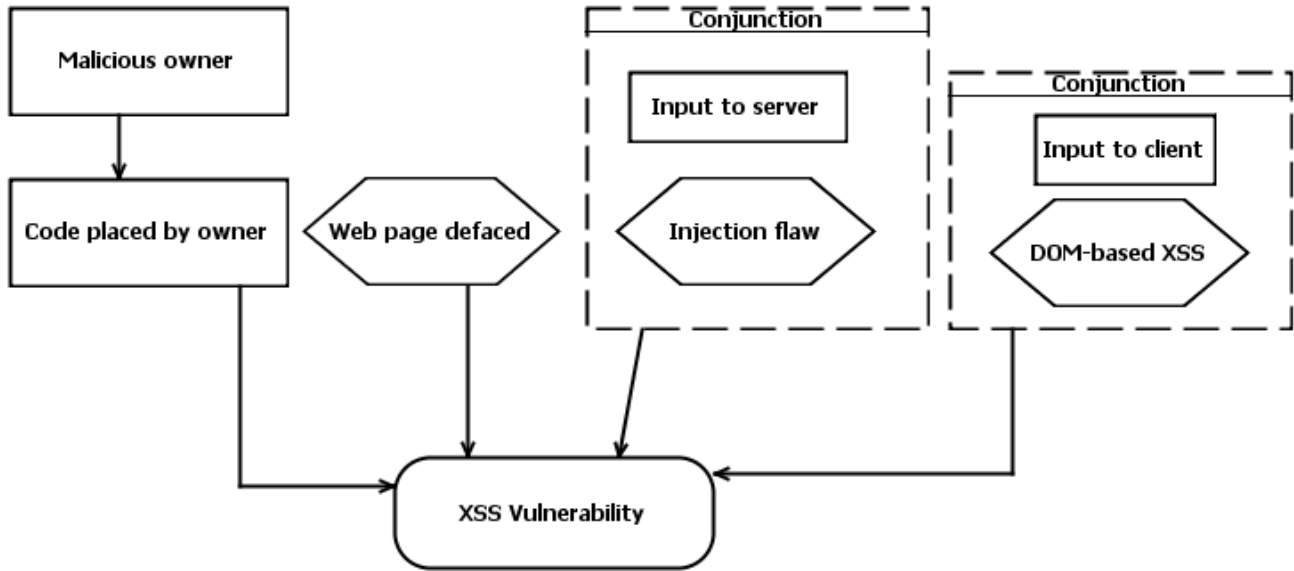


Figure 1. A vulnerability cause graph for the XSS vulnerability.

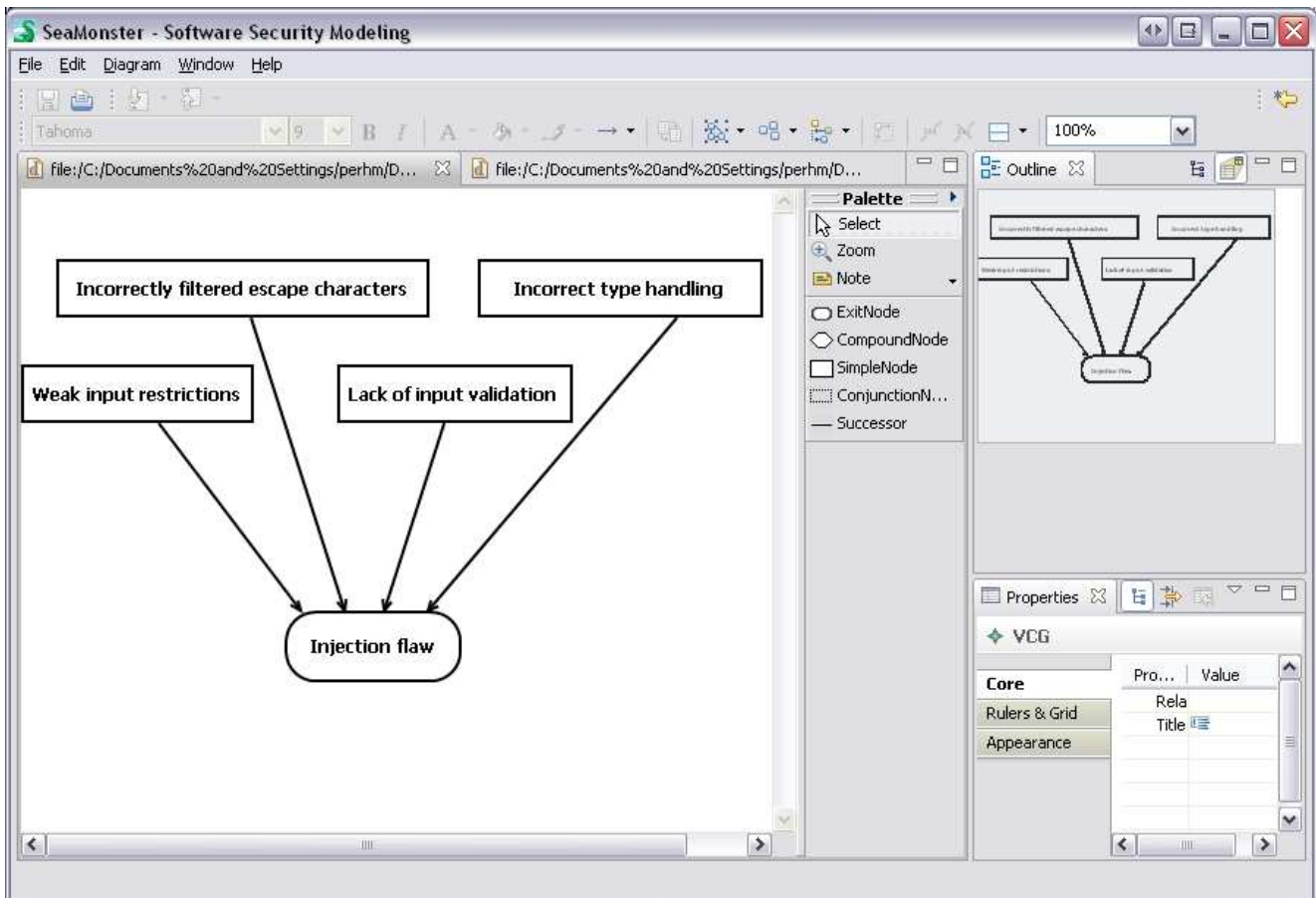


Figure 2. Screenshot from SeaMonster showing the details of the injection flaw vulnerability.

some code into the Web site. This is expressed with two simple nodes. In the next case, we express with a compound node that the Web page has been defaced, typically by gaining root access to the Web server and replacing the content with malicious code. The third option is shown with a conjunction node around a simple node and a compound node, meaning that both need to be present. Here, the Web server accepts some input, typically from a Web form, and an injection flaw vulnerability is present, allowing malicious code to be injected into some public area of the Web site. A similar construction is the last option, but there the user can be tricked into clicking on a maliciously crafted link (the malicious code does not need to be echoed by the Web server). See [6] for a more thorough explanation of these causes.

The injection flaw vulnerability is further detailed in Figure 2. In this figure the modeling environment can also be seen. SeaMonster allows us to combine and decompose vulnerabilities using the compound nodes, keeping the graphs to a limited visual size.

VCGs give a good overview of the relations between a vulnerability and its causes. They support generalization and specialization so that models can have different levels of details, and also the fact that a vulnerability often is the result of a combination of causes, not just single ones.

## 4 Threats and attacks

Threats and attacks consider the weaknesses of a system from either a defender or an attacker's perspective. The modeling methods used to represent this is usually threat or attack trees. We have merged these into the same view because they have the same notation and tend to get mixed up into each other anyway (you usually try to think as *both* a defender and an attacker when creating the tree). Both of them are closely related to fault trees, but more specialized towards security. Modeling threats and attacks is something typically done early in the development phase, when you want to analyze the attack surface and identify the risks and consequences of potential attacks.

In SeaMonster we use attack trees as described by Schneier [14]. Attacks are represented in a tree structure. The root node represents the main goal of the attack, while the successor nodes represent ways to reach the main goal. In this way, all child nodes become subgoals. An example of a XSS attack tree modeled in SeaMonster is shown in Figure 3.

Just as fault trees, attack trees use *AND* nodes and *OR* nodes. The first node type requires all subgoals to be satisfied in order to reach the goal, while the latter is used to describe alternative ways of achieving the goal.

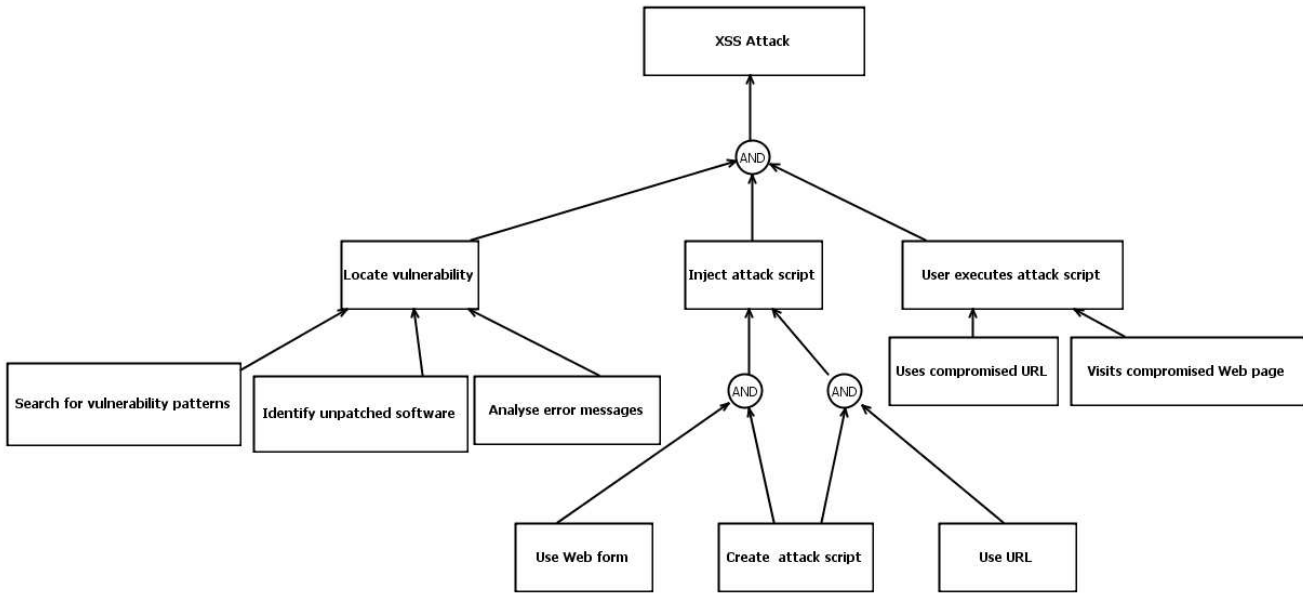
The nature of the attacker may determine which part of the attack tree should be taken into account. It is therefore important to consider who the possible attackers are (what skills they may have, risk aversion, access to money, resources, motivation and so on).

Attack trees are simple, informative, and relatively easy to understand. These are valuable properties when communicating to a diverse audience.

## 5 Countermeasures

The countermeasure viewpoint considers how to avoid vulnerabilities in the first place, or at least mitigate possible attacks or threats that are caused by them.

UML use case diagrams are widely used to elicitate software requirements, but they are not that good when it comes to security requirements. By adding new types to the use case diagram, security issues can also be addressed. This extension of the use case diagram is called either misuse cases [17] or abuse



**Figure 3. An attack tree for exploiting the XSS vulnerability.**

cases [10] (we have chosen to use the former). The purpose of the misuse case diagram is to model the desired functionality and relate it to exploitable functionality and threats.

Use case diagrams have four major elements. These are the *system*, *actors* using the system, the *functionality* of the system (use case), and *relationships* between the different elements. The relationships are limited to *include*, *extend* and *generalization*. The model is extended for security issues by adding an attacker [17] and/or an insider [13]. The attacker is a mis-actor that attacks the system from the outside. The insider is a mis-actor that is authorized within the system, but uses this advantage to exploit vulnerabilities. The model is further extended with three relations: *detect*, *prevent* and *exploit*. The exploit relation is used between the threat and vulnerability use cases. The other two relations are used by regular use cases to mitigate the threats and vulnerabilities. The mis-actor is colored in black, the vulnerability and threat use cases are shown with different grey shadings. The insider has also a grey shading. See Figure 4 for a XSS vulnerability example modeled in SeaMonster.

The misuse case diagram is not a complete method for documenting and analyzing security requirements, but it provides support when working on these tasks. The misuse case diagram can get very extensive when covering all security details. Although it should only be used for doing a part of the job, it can be a good way of getting an overview of countermeasures. The misuse case diagram is also a good way of communicating security requirements between security experts, developers and customers.

The security activity graph [3] is another view describing which activities are available to eliminate threats at various costs. The security activity graphs can be constructed from vulnerability cause graphs. The causes in the vulnerability cause graphs are mitigated using different techniques, that corresponds to activities in the security activity graphs.

The top node in the graph contains the vulnerability, and the leaf nodes contain the activity that can help prevent the vulnerability. The activities have boolean values, which are true if the activity is implemented perfectly. The activities can be connected together with *AND*, *OR* or *SPLIT* gates. *SPLIT* gates only have one input, and they send this input to all the outputs. An example of a security activity



Figure 4. A use case model extended with misuse case notation to represent the threats and countermeasures of a XSS vulnerability.

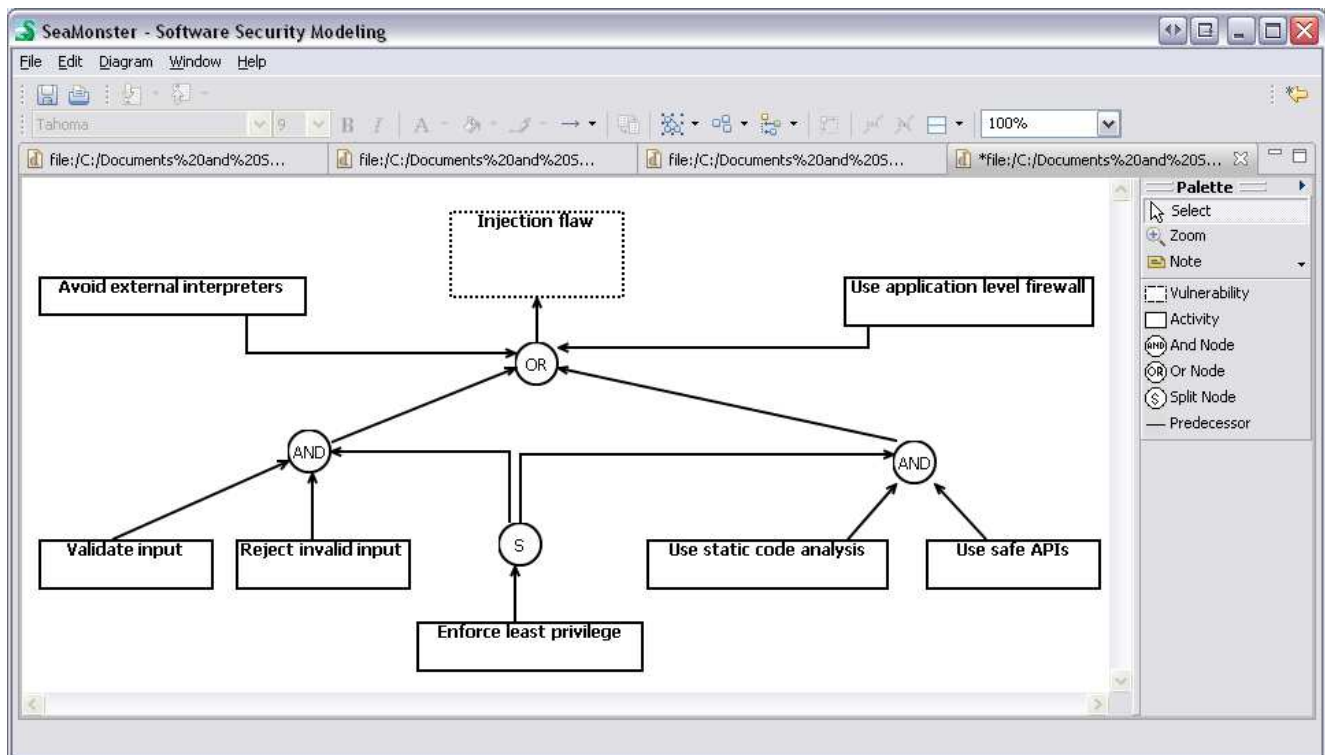


Figure 5. A security activity graph for the typical injection flaw.

graph for mitigating the injection flaw vulnerability is shown in Figure 5.

Compared to misuse case diagrams, security activity graphs are more detailed and process oriented, and the view should be chosen according to your needs. Security activity graphs can typically be used to guide developers during design, implementation and testing, but should be created outside of specific development project by security experts.

## 6 Discussion and further work

As we have already mentioned, most security modeling is done using the available general purpose drawing tool (or temporarily written on the nearest blackboard). We want to change this trend by making a common platform for modeling security. Our goal is to increase the general understanding of vulnerabilities and reduce the amount of time it takes to model security. We hope to achieve this by facilitating exchange and reuse of security models through SeaMonster. A long term consequence should be a reduction in the amount of vulnerabilities in released software and less need for patching.

Some specialized tools already exist<sup>3</sup>, but they tend to support only a single view and are not freely available.

Microsoft is an important actor in the field of threat modeling. Their approach to threat modeling is explained in the book [19], which is complemented by a threat modeling tool that is freely available from Microsoft. However, this tool only has a view that supports textual descriptions of threats. This kind of modeling could prove useful for development teams in order to identify both security strengths

<sup>3</sup>Such as SecurITree by Amenaza and Isograph's AttackTree+.

and weaknesses of a system, but due to its high level of detail it could need more time than available, and be considered too complex and not feasible for most software projects.

Another visual and free modeling tool is Coras [18]. Coras is used for model-based risk analysis of security critical systems, but has an irregular syntax that reduces the uptake.

Exploit graphs are used in [12] to model how an attack can be carried out, given a software risk. It also describes what kind of access and/or pattern is required to do so. Regular flowcharts are used to model the exploit graphs. The graphs include an estimation of the effort level required to exploit a flaw in the application. However, exploit graphs are not widely used and have therefore not been added as a view to SeaMonster at this point.

UMLsec [8] is a UML security modeling method, with the goal “security by design”. The UML standard is used because it is commonly known by software developers. UMLsec is an extension of the UML standard and it includes, among others, use case, activity, deployment, sequence, and state charts diagrams. SeaMonster already supports misuse cases, and we see them as the most beneficial UML extension when it comes to security vulnerability modeling (note that SeaMonster is not intended to be a full-fledged design tool, but merely a security companion).

SeaMonster has been created so that more views and functionality can be easily added. For instance, the attack trees should be extended to include a cost for each node. At an *OR*-gate choose the cheapest child, at *AND* nodes sum the children. This could further be used to calculate the total cost of the different attacks, and by doing this, figure out which attacks are the most likely. In this way, one can rule out the less probable attacks. Also, we would like to add functionality for linking external resources to the elements in the various graphs, such as URLs to vulnerability information, other tools, patterns, practices etc.

Ardi et al. [2] discuss sharing of security models, and this is functionality that would benefit SeaMonster greatly. A common repository for sharing and reusing good security models is currently under development in the SHIELDS project [16] supported by the European Commission in the Seventh Research Framework Programme. We hope that SeaMonster will be able to take advantage of this repository in the future.

## 7 Conclusion

Security experts are often not a part of development teams, and software developers have little security knowledge, so a gap between the two groups arise. Security models can be used to exchange knowledge between these two camps, however proper tool support is vital to achieve this. SeaMonster is such a tool, and uses well-known and easily understandable security modeling notations and techniques. By providing and linking different viewpoints to vulnerabilities, different aspects can be expressed, such as causes, threats and countermeasures within the same tool.

SeaMonster is a freely available open source project, and we hope it will contribute to the security modeling field by facilitating model reuse and improving the general security-awareness in software development projects without adding too much extra cost.

## 8 Acknowledgments

The authors would like to thank Eirik Benum Reksten for contributing to the initial implementation of SeaMonster. Guidance and support from Jostein Jensen, Renate Kristiansen and Basit Ahmed is

greatly acknowledged. We will also thank Ole Gunnar Borstad for his security activity graph addition to SeaMonster.

## References

- [1] B. Andersen and T. Fagerhaug. *Root Cause Analysis: Simplified Tools and Techniques*. ASQ Quality Press, 2006.
- [2] S. Ardi, D. Byers, P. H. Meland, I. A. Tøndel, and N. Shahmehri. How can the developer benefit from security modeling? In *ARES '07: Proceedings of the The Second International Conference on Availability, Reliability and Security*, pages 1017–1025, Washington, DC, USA, 2007. IEEE Computer Society.
- [3] S. Ardi, D. Byers, and N. Shahmehri. Towards a structured unified process for software security. In *SESS '06: Proceedings of the 2006 international workshop on Software engineering for secure systems*, pages 3–10, New York, NY, USA, 2006. ACM.
- [4] D. Byers, S. Ardi, N. Shahmehri, and C. Duma. Modeling software vulnerabilities with vulnerability cause graphs. In *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 411–422, Washington, DC, USA, 2006. IEEE Computer Society.
- [5] N. Davis. Developing secure software. *The DoD Software Tech News*, 8, 2005.
- [6] J. Grossman, R. Hansen, A. Rager, P. D. Petkov, and S. Fogie. *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress, 2007.
- [7] M. Howard. Building more secure software with improved development processes. *IEEE Security & Privacy*, 2, 2004.
- [8] J. Jürjens. *Secure Systems Development With UML*. Springer, 2005.
- [9] P. Kemper and W. H. Sanders. *Computer Performance Evaluation: Modelling Techniques and Tools*. Springer, 2003.
- [10] J. McDermott and C. Fox. Using abuse case models for security requirements analysis. *ACSAC*, 0:55, 1999.
- [11] G. McGraw. From the ground up: The dimacs software security workshop. *IEEE Security and Privacy*, 1(2):59–66, March-April 2003.
- [12] G. McGraw. *Software Security: Building Security In*. Addison-Wesley Professional, 2006.
- [13] L. Røstad. An extended misuse case notation: Including vulnerabilities and the insider threat. In *The Twelfth Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'06)*. Essener Informatik Beitrage, 2006.
- [14] B. Schneier. Attack trees - modeling security threats. *Dr.Dobb's journal*, 1999. <http://www.ddj.com/architect/184411129> Last date accessed 2007-09-22.
- [15] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. Wiley, 2006.
- [16] SHIELDS consortium. *SHIELDS - Detecting known security vulnerabilities from within design and development tools*. <http://www.shieldsproject.eu/> Last date accessed 2008-07-09.
- [17] G. Sindre and L. Opdahl. Eliciting security requirements with misuse cases. *Requir. Eng.*, 10(1):34–44, 2005.
- [18] SINTEF, Telenor. *The CORAS UML Profile*. <http://coras.sourceforge.net/> Last date accessed 2007-10-26.
- [19] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft Professional, 2004.
- [20] The U.S. Commerce Department. *National Vulnerability Database web page*. <http://www.nvd.nist.gov/> Last date accessed 2007-09-24.