Introduction to Modeling Tools for Software Security

Samuel T. Redwine

February 2007

ABSTRACT: Identifying security properties is a non-trivial undertaking within system and software requirements and design. Certain tools can aid in the modeling and analysis effort required. Some of those tools were created with security in mind; many were not but are, nevertheless, useful. This article introduces the security concepts and the kinds of tools that are useful for modeling security properties.

INTENDED AUDIENCE

While students and others may find it a useful introduction to the subject, this article is primarily intended for practicing specifiers and designers of software systems and their managers. It presumes a general familiarity with software and to a lesser extent security. While this article does not presume a background in the modeling of software, the General Modeling Concepts article in this content area provides general information about modeling that may give a richer understanding of some content. A few parts are easier to understand if one has a general knowledge of UML and dataflow diagrams.

Those who have a sound background in modeling of software and of software security may still find this article useful in recalling and organizing the many concepts and issues involved.

SCOPE

The scope of this article is limited to analysis, specification, and design tools relevant to software security. The modeling tools of interest deal with one or more artifacts from concept descriptions and specifications through detailed designs. Many deal with system properties, security knowledge or principles, or composition/decomposition. Security requirements and emergent system properties are, of course, of interest, as are combining software entities (e.g., components) so that they provide or preserve the required security, and tools that address other security issues or analyses.

Software Engineering Institute
Carnegie Mellon University
4500 Fifth Avenue
Pittsburgh, PA 15213-2612

Phone: 412-268-5800
Toll-free: 1-888-201-4479

www.sei.cmu.edu

Given its limited length and the need for wide coverage, this article's attention will necessarily be restricted to categories of tools with mention of a few examples. The paper has, however, pointers to lists of tools and/or vendors published in print and/or on the Internet.

Unfortunately, because attackers might eventually exploit almost any weakness in a system, a clear division between security weaknesses and other kinds of design faults is not always possible. Furthermore, historically some design practices have not been widely recognized as resulting in security weaknesses until they were exploited. Some areas of design difficulty such as concurrency including distributed asynchrony [Damianou 2002a], however, obviously have more potential for generating security problems than others.

While including a broad scope of types of analysis-, design-, or specification-related tools, the tools addressed elsewhere on the BSI site are avoided. Thus, tools primarily addressing code, threat modeling, risk management, assurance cases, protocols, cryptology, or testing receive less coverage than they might otherwise. However, this area includes some tools that create or input design information reverse-engineered from code and some design tools that generate code or are otherwise intimately connected with design.

Security

While externally system security is concerned with illegitimate or malicious actions and their consequences, internally it is generally concerned with confidentiality, integrity, availability, and accountability regarding sensitive computing assets. It also relates to assurance—reducing uncertainty and providing grounds for justified confidence. Issues of intellectual property, protection of the software, non-repudiation, anonymity, and compliance with regulations and other governing documents are included. (See the BSI article Introduction to Software Security for further details.) This article focuses on modeling the software system's consistency with its security requirements and violation-related modeling such as of attacks, weaknesses or vulnerabilities, violations, and the risks and consequences thereof.

More security modeling has been done related to confidentiality than integrity. Much safety modeling is labeled as being related to "integrity," but the word is often not used the same way as in security. While safety-oriented modeling techniques often are useful or suggest analogous techniques for security, care must be taken concerning terminology and possibly inappropriate use of probabilistic analyses.

Specifications of security requirements and rules are sometimes called security policies. Policies must consider the software's security over its lifespan, including anomalies such as computer or media theft.

Ultimately, security exists to limit damage and provide peace of mind, but in practice, it must also comply with governing directives including regulations, standards, and organizational policies.

Organization of this Article

This article has three major parts:

- Security-Relevant Modeling in Software Development: important subjects and objectives for the use of modeling across the life cycle, including security objectives, properties, and aspects and their modeling. This section includes subsections on requirements, design, and assurance cases.
- Tool Characteristics: This section explains the general nature and relevance of key tool characteristics.
- Selecting Tools: factors to consider when addressing the selection and use of tools to aid in software-security-related modeling

The article ends with a summary section. References are given for those who wish to learn more about the topics, methods, notations, or tools mentioned.

SECURITY-RELEVANT MODELING IN SOFTWARE DEVELOPMENT

Modeling tools are used throughout the development life cycle to support the creation and modification of secure software and analysis and verification. Security-relevant modeling tools are primarily used for discovering particular kinds of faults, achieving correctness, and aiding the creation of or checking conformance with security policies or specifications.

These tools may also incorporate aids and specialized knowledge that strictly speaking might not be labeled "modeling" but that support related tasks. These may include lists or catalogs of potential parts of an artifact or system and checklists or other aids for human performance of reviews. An example of such a tool is the EBIOS tool, which aids in security requirements analysis and definition.

Requirements

While general tools relevant to software system requirements can be used to record and perform some analyses, several specialized tools also exist. Modeling

requirements often involves describing or modeling aspects of the software system's environment. This section addresses modeling and tools for threat analysis, creation and compliance with security policy, specification (of functionality, for example) and its consistency with policy, and tools with broad coverage of security requirements activities.

Threat Analysis

Threat analysis is a source of security requirements. Threat analysis addresses such issues as possible threatening entities or categories of attackers, their present and future capabilities, and possible consequences from their actions. (The latter, as well as analysis of designs for vulnerabilities, will be discussed more under Design.) Some tools combine threat analysis and vulnerability analysis. An example is the Microsoft Threat Modeling Tool (available through the Microsoft Download Center).

Compliance with Governing Documents

Government regulations such as Sarbanes-Oxley have had a significant influence on security requirements. In large part, security policies for a software system derive from societal mandates (e.g., laws and regulations), organizational policies, and larger encompassing system policies, as well as the effects of interfacing systems. While in practice often roughly half of the compliance effort deals with non-technical issues such as work rules and processes, the software system's security policy must meet all compliance requirements relevant to its behaviors and must include the particular policies specific to it.

Several tools help organizations create and realize security policies that comply with standards and U.S. laws and regulations. These include specialized tools from Polivec, Symantec (BindView), and Zequel. Less-specialized tools from Archer Technologies, Cybertrust, McAfee (Preventsys), and NetIQ include this functionality.

Constraints on System Behavior

Security requirements can also be internal. The design of a large system describes the security constraints of its components. These security constraints reflect the software's responsibilities towards relevant policy principally for the prevention of, avoidance of, or accountability for violations. Many of these behavioral constraints are all encompassing, as in "No behavior of the software shall result in X."

In addition to constraining otherwise existing functionality, these constraints may imply the need for added functionality, such as a constraint on who may do what (implying a need for functionality to adequately identify people). In turn,

this may require related constraints, such as not allowing bypassing of this identification functionality (e.g., of authentication).

Because of its special concern to software developers, these modeling tools often chiefly consider security policy as it directly constrains the behavior of a software system. These constraints often relate to entity identities and constraints to exist under certain conditions or to doing various operations or actions. Most of these constraints are reflected in access controls that during operations will be driven by user-specified rights or privileges. Common relevant topics include identification and authentication, authorizations, and logging and auditing.

The security properties specified in the security policy as constraints on behavior must be compared to the system's specifications or designs that describe the functionalities' behaviors. In the face of this, to make the verification of the system specification's compliance with its security policy practical, both must be unambiguous, talk about the same entities and operations, use the same or interoperable notations, and have comparable levels of abstraction.

During operations, security policy enforcement deals with specific individual or categories of entities and actions and are defined by constraints on actions because of the actor's, action's, and action target's categories and attributes as well as possibly the context and the conditions that are currently true. Thus, they can involve much more than just the identity of the actor and the type of action. The actor "accesses" its target in order to perform the action—say a user accessing a file (or, more accurately, on behalf of the user some process accesses the file system, which accesses the file). An access control policy is generally dynamic. Identities may be established or removed and their privileges (and obligations) granted and revoked, and assets may change. The software system's requirement is to provide a means to manage and perform policy definition or access control administration, enforce the policy while ensuring accountability, and handle violations of policy while itself never being the source of security problems.

Security Policy Model

Security policy model is a traditional name for the combination of the

- specification of the security policy—normally constraints (or properties)
- specification of the behavior of the system—normally a high-level specification of the design1
- argument showing the consistency of the two—normally this means showing that a software system always stays within security constraints

This need to show consistency has been already been mentioned. In the early days (1960s, 70s, and 80s), this was often written about in terms of formal

proofs. In the 1980s the concept of levels of assurance mapped to different kinds of evidence for this consistency. The legacy of this work is seen in today's Common Criteria. However, borrowing in part from experience in safety, this concept has been generalized to one of an assurance case, which in part tries to address its own uncertainty and is intended to provide grounds for justified confidence and decision making by stakeholders.

One example of tool use is Praxis's use of Z/Eves to formally state security policy and show the consistency of the system's high-level design with it, using mathematical logic [Hall 2002].

Security Requirements Tools with Broad Coverage

EBIOS is an example of a tool that attempts to provide support for security-related activities from requirements elicitation and threat analysis through policy and the identification of required functionality. While its modeling aspect is quite limited, its embedded security knowledge, including techniques, artifacts, and possible requirements and functionality, is extensive. It is accompanied by a five volume guide to related methods and techniques that has merit even apart from using the tool.

In using security-oriented tools with embedded knowledge concerning security functionality, ensure that security properties and security functionality are not bypassable. The existence of the needed security functionality does not by itself guarantee security.

Design

Showing consistency of design with requirements forms the bridge between requirements and design. The security policy model discussed above is one approach. A number of other issues and models also relate to the security aspects of design.

The identification and specification of the functionality needed to carry out the security policy and meet security requirements is a key issue in design. Tools may provide lists or catalogs of possible functionality; they may also provide guidance and analysis. The example of EBIOS's support for this activity has already been mentioned.

Security aspects of your design should facilitate or at least minimally hamper the software system and its users doing their legitimate tasks while minimizing the opportunities for illegitimate actions and the size of losses as well as coping with their occurrence and aftermath. Users like flexibility, so you want to accommodate them as much as possible. On the other hand, making certain security functionality non-bypassable is normally an inviolate requirement of the system.

Models providing measures of merit for the amount and kinds of interaction paths include simple counts and more elaborate formulae, for example [Manadhata 2004].

Modeling authorization and authentication can generate a spectrum of models:

- Business process and role models: Role-based access control often fails because we cannot consistently model the roles in an organization.
- Policy model: A privacy or confidentiality policy may have to follow the data as it moves within and among systems.
- Performance model: What is security's impact on system performance, and does the design scale?
- Data management model: Changes in permissions may require synchronization of user security information across multiple systems.

This ability to make changes and do proper administration is critical. The system may be implemented correctly, but the identification and authorization data also needs to be maintained correctly.

The modeling of inference is usually difficult. This is the problem of what an entity might infer not from the direct disclosure of something but from legitimate disclosure of other information or behavior of the system that is visible to the entity. Driven by privacy concerns, considerable work exists on this problem in databases that provide only statistical or group results and what might be inferred about individual values in the database. You should take advantage of this if it is relevant to you but recognize few definitive results or models exist.

Adaptations have been made to existing modeling approaches and notations to facilitate addressing security, usually accompanied by some adaptation of an existing tool. Two examples for UML are described in [Jürjens 2004] and [Lodderstedt 2002]. A number of security-oriented design patterns have been created as models for portions of designs and their use addressed [Fernandez 2007]. Aspect-oriented design is another emerging approach [Viega 2001].

State machine modeling approaches have also been adapted. Automated verification of state machines has advanced over the last 15 years. Model checking tools for state machine models have become surprisingly powerful and are able to handle enormous numbers of states. Commonly used tools include SVM and SPIN. The most common uses for model checking are concurrency and protocol analyses. Despite the huge number of states that modern tools can handle efficiently, abstraction of a design to have fewer states is often needed. Even so, analyses to establish that the system will never enter a state that violates security are not always carried to conclusion. A breadth-first exploration of reachable

states might be done to identify all states that can be reached from its starting or initial state in say ten state changes.

State machine models have also been used to generate tests, including ones trying to violate security. An example of this in the software security area is described in [Blackburn 2001].

Lack of Weaknesses or Vulnerabilities

While creating a design using sound methods, principles, and positive verification of required properties and functionality is essential, good engineering calls for also looking for known pitfalls or other weaknesses in the design. Attack and vulnerability analyses are the two main activities where modeling is done to discover and fix or mitigate weaknesses or vulnerabilities. Dangerous events and their causes and consequences are subjects for modeling.

Not all vulnerability seeking tools are modeling tools. Some simply scan for weaknesses or conduct and analyze tests.

Increasing the difficulty in this area is the frequent inability to justify the use of probabilistic models. This is due to the misleading and perverse behavior of attackers who tend to attack where, when, and how you least expect. Nevertheless, a number of approaches assign probabilities to different attack possibilities in order to categorize the dangerousness of a given weakness.

Modeling of covert channels is another specialized area. Covert channels are abnormal means of communication from computing areas with higher secrecy levels to lower ones. The timing of overt messages, the use of message bit locations not normally used, or availability of resources may convey these messages covertly. Covert channels are modeled to determine the bit rate that they can pass—maximum or average.

Modeling Dangerous Events

Of course threat analysis has an influence on security policy, but when modeling for discovery of potential weaknesses the emphasis is often on actions that an attacker might take or the implications of partial or complete success of attacks. One important kind of implication is the consequences that could result from security violations and possibly mistakes in policy or usage.

Modeling using attack trees and attack patterns is employed to try to find vulnerabilities or weaknesses. Attack paths can be modeled via these methods or by other modeling techniques. These attack-oriented models may simply record human review and postulation, derived from catalogs of attacks, or ones generated automatically from the design. In addition, Microsoft has used Dataflow Dia-

grams to help identify attack paths. Example approaches include those in [Swiderski 2004], [Steffan 2002], [Liu 2005], [Liang 2005], [Hoglund 2004], and [McGraw 2006].2

Potential cause and root cause analysis can be used to model attacks by working backwards from consequences to triggering events and prerequisite or prior conditions, and by identifying contributing causes determine the best response. Experience in modeling of accidents within the safety community indicates this is often efficacious, but sometimes it is not chains of events that cause accidents but inherent behaviors of complex systems that are acting "correctly." Complexity is also a difficulty in security.

Modeling Consequences and Risks

Models of risks and consequences related to violations and failures given the design are important for design. Probabilistic risk modeling and tools have a long history in finance and in safety. You may find ones applicable to parts of your problem, and some of these techniques have been applied to security, such as [Schechter 2005]. Design issues such as avoiding single points of failure or single points whose violation or capture would result in total or severe consequences can be modeled.

Consequences can be modeled and designs can attempt to limit or reduce potential or real negative consequences of violations by, for example, limiting the set of entities or resources that are at risk (inside or outside the system) or limiting or excluding assets. You can model to evaluate how well you have limited various consequences in size, location or propagation, time or duration, nature, or degree of surprise or unpreparedness. Tolerance for attacks, including recovery and repair, are important areas for modeling.

The use of an assurance case (discussed below) and risk management modeling tools and techniques can be relevant here as well.

Software Security Assurance Case as a Model

Models appear within an assurance case in two roles. The first is as part of the evidence and the second is as bases for arguments that support the security claims of consistency with policy by linking supporting evidence to the claims. An example of an assurance case tool useable for security is the Assurance and Safety Case Environment (ASCE) from Adelard.

The requirement for composability is central to achievement of security, and showing that the system has security property preserving (de)composition is an essential element of the assurance case. At the highest level a model can show that the design's behavior agrees with the security policy. At each level the de-

composition or refinement of a part of the system into its next lower level parts and the way they relate to each other must preserve the security properties of the higher level part. Thus, one needs a way to reason about this—a model of that portion of the design.

One also needs the design to be appropriately modeled whenever you wish to show that the security properties can be achieved by trusting just a subset of the software.

Thus, while it contains other content, essential portions of an assurance case often rely on design models. In addition, the assurance case as a whole can be thought of as a risk-oriented model of the software or system.

Future Trends

Several trends and possibilities exist in design modeling. Probably the most talked about one is Model Driven Architecture (MDA) or more generally model driven development or engineering (MDD or MDE), one of whose key ideas is generation of software from design descriptions. Another trend is component-based development (CBD), a third is web services, and a fourth is systems of systems (SoS).

While additional specialized security modeling tools may be developed to address these, the greatest need is to incorporate quality attributes such as reliability and security into more general tools despite the considerable difficulties in doing this. One example is Microsoft's Software Factory attempt to model quality attributes.

MDA or MDD has received considerable recent attention, particularly from the Object Management Group (OMG),3 but also from other framework or standards organizations and vendors. Tools supporting description of architectures or designs and exchanging these descriptions have proliferated.4 In addition, the OMG is producing exchange standards for exchange of relevant data among tools—something that may have even more future impact. As these tools begin to address security, the reliability and trustworthiness of the tools themselves will become even more serious issues.

TOOL CHARACTERISTICS

This section provides short descriptions and discussions of important characteristics to consider for software design modeling tools. These cover a tool's functionality, usage, quality, and methods used. Some characteristics may relate more to the tool or to their embedded modeling approach. For example, scope is often

more a modeling issue, while scalability is more often limited by the implementation of the tool.

Scope

A tool or toolset and its modeling capabilities may cover more or less of the life cycle and aspects of design. It may or may not include networking or cryptographic concerns. It may be suitable for some subset of security properties, say most of confidentiality, but not accountability or intellectual property protection. Uncertainty may be expressed explicitly or ignored. Some tools specialize in particular application domains or level of abstraction. All of these and other such scope differences can significantly affect the suitability of a tool for your particular project and purpose.

In addition to suitability for current uses, the tool's ability to do everything the organization anticipates needing in the future can be important to technology adoption decisions.

Conceptual Basis

A number of conceptual bases have been mentioned, such as logic and set theory, state-machine-based reachability analysis, access control models, inference and noninference, attack trees, and categories of weaknesses. Probability theory and risk management models are also possible bases. A given tool will have one or more underlying bases for its models and how to reason about them.

Notations Used

Although a tool may have different external and internal representations for its models, the notation(s) used by tool users place limitations on the descriptive power and the rigor and kinds of analyses and results it can provide. However, explicit user-visible notation may not be the only input to the tool; it may also gain input from other existing information. Its notations also have effects on the acceptability and usability of a tool.

Level of Rigor

Informality can aid accessibility, but it lends itself to ambiguity and is a serious impediment to deep or precise analysis. Structuring information with semiformal notations involving such means as tables and diagrams can aid understanding and increase usefulness. Tables and diagrams can also be used to express formal notations, as can more traditional means such as mathematical notation. The level of rigor is a key determinant of what analyses the tool might perform.

Amount and Nature of Security-Related Knowledge Included in Tool

While recognizing that no tool is perfect or necessarily fits your situation, embedded expertise can be of considerable help. Some tools such as EBIOS and the Microsoft Threat Modeling Tool contain knowledge and techniques that can automatically bring security expertise into your security-related creation, modeling, and analysis tasks. This is not to recommend these two tools but to point out the possibilities and advantages of tool-supplied security expertise.

Analyses Provided

Many different capabilities are available in different tools. These may include simulation and measurements as well as analyses. A few examples of analysis capabilities are

- syntax and type checking
- coupling and cohesion measurements
- exception identification
- checking functions and relations for empty domains
- information flow analysis
- reachability analysis
- precondition calculation
- general theorem proving
- reverse engineering of source code
- analyses of simulation results

These analysis capabilities combine with the scope of the tool and notations (and their rigor) to determine much of the power of a tool.

On the other hand, like all software products, modeling and analysis tools can suffer from having too many unneeded features (featuritis) and become overly complex, hard to learn, and too expensive.

Soundness and Completeness

Often an important question is Is everything an analysis reports true, or does it also report likely or suspicious items? Another important question is Does it always report all of the items of a certain kind, or are its results possibly incomplete? The answers to these questions can make significant differences in the level of uncertainty that its analyses leave you with.

Scalability

How big a design or system can the tool handle? Can this be readily increased as needed? Software in general is increasing in size, and abstracting or reducing the

size of the (pretend) system so your model will fit within a tool's capabilities can be difficult and weaken confidence that nothing essential to the analysis has been left out. Given the investment in a tool and the skill needed to use it, some time may be required to rebuild the model when the software increases in size. Thus, scalability is often a significant issue, including in user acceptance of the tool.

Acceptability and Usability

Is the tool acceptable, or even though technically desirable does it, for example, scare people or use ugly colors people hate? What prerequisite skills are needed to begin learning how to use it? How easy is the tool to install and learn? Is it slow, mistake prone, hard to remember how to use, or unsatisfactory to its users? On the other hand, are the complexities of the tool and its analyses invisible to the user and take input the users are already producing? These kinds of factors can make the difference between a successful tool or design model and an unsuccessful one.

Compatibility with Other Tools

Today, no single tool or toolset has all the capabilities one might desire for describing and modeling designs for the six purposes mentioned earlier. Therefore, being able to use multiple tools in effective combinations is necessary to gain the maximum advantage from design modeling. The low cost way to do this is for the tools to already be compatible.

Correctness and Security of Tools

All modeling tools have potential for defects and the inclusion of malicious software. Design tools that generate code usually produce fewer faults in the code than the average programmer does. Nevertheless, a code-generating tool might or might not result in high-security code, so unless this is an explicit goal of a highly reliable tool, the tool may have difficulty providing adequate input to your assurance case. In theory, tools should have at least as good an assurance level as that intended for the software being produced using them. If this is not true, using a results checking tool (e.g., a proof verifier) that is small and easier to gain confidence in (always a good idea) can be particularly valuable. In addition, using and comparing the results from multiple tools can mitigate problems. However, it is essentially impossible to determine how much mitigation this provides.

Code Generation

Not all code generation is the same. Does the tool generate complete code ready to run? If not, how much or which portions of the code does it provide? Is the code produced source code, an intermediate object code, or the final binary?

Does it provide make files or tests to accompany the code? (Beware the same tool faults causing problems with both outputs.) And finally, what non-generated software does it depend on to execute, such as libraries, database products, or frameworks?

Test Generation

In addition to code generation, test generation is another example of a tool's ability to map from the more abstract to the more concrete. Automated test generation from specifications and designs has been done in a number of situations. Such test automation is not a complete answer to testing but may substantially aid speed and efficiency.

Forms of Results

All of the characteristics listed so far can have effects on the kinds of results or outputs a tool provides. But other dimensions exist as well. Does it provide summaries, indications of changes, graphs, denoting by color and shape, output selected or sorted by measures of importance or relevance, show the output alongside the input that produced it, version identification, history of prior analyses, traceability, file types or exchange formats suitable for input from or by other software, etc.? All of these can make a difference depending on your situation.

Other

As with all decisions, the better you understand your needs and possible solutions, the better your tool selection decision is likely to be. After analyzing your situation and needs for design or specification modeling tools, you may find characteristics not on this list that are important to you. However, you should keep in mind the ones listed here, as overlooking them when they are important to you could lead to a poor decision.

SELECTING TOOLS

If possible, tools should not only provide straightforward features but facilitate dealing with key concerns. For example, verifiability and your software's assurance case need to be considered from early conception and planning for the software's development.5 In addition, simplicity and other security design principles, such as minimizing the amount of software that you must trust, can ease these concerns. While this is important to keep in mind, straightforward concerns and basic features may dominate your decisions. These include notations, creat-

ing and editing artifacts, the kinds of analyses provided, and fit with your software life-cycle processes.

1. You should ask yourself at least these questions:
2. What are your objectives for using the tool?
3. Where in the life-cycle activities will it be applied?
4. Is the tool consistent with and supportive of your life-cycle process? For example, what are the artifacts used and developed and how are they managed?
5. What is the model or modeling approach that is the basis for the tool?
6. How effective is that model for meeting the desired objectives?
7. How well does the tool support the model?
8. How usable is the tool?
9. How acceptable to your organization will the tool be?
10. Do the tool and/or your processes need to be adapted for it to work in your organization?
11. How easy would it be to introduce it into your organization and have it successfully used?
12. How can you best use what is known about technology transfer and organizational change to make the tool successful in your organization?
13. How will you know its level of success or if you need to later change your decision?

The stimuli and objectives for adopting security-related software tools vary. You may aim to produce a better original requirements or design artifact or you may want to find and possibly fix weaknesses in the artifact. You may desire to validate your system's specification to ensure you have specified a system that will be successful and adequately secure in the real world, or you may want to verify the specification by comparing artifacts (e.g., security policy with system design) for consistency or "correctness." Some of your objectives may aim at security and some at assurance, or you may want both at once. In addition, you may wish to increase efficiency and efficacy.

Whatever your specific objectives, seriously addressing security requires changing existing processes that do not do so. Likewise, achieving high assurance would necessitate changing most organizations' processes. Thus, you may need to change your processes or methods not just to best utilize a tool or tools, but to achieve more secure software. Thus, the tool's fit to your process may not be simply how well it fits your current process, but how well it will aid you in your revised software process.

You should chose notations that have the tool support you need. However, more than security-oriented tool support goes into decisions concerning notations. Factors such as their understandability by key stakeholders, tool cost, amount of training needed, and ease of use influence which notation you choose.

Ease of editing is important, but for substantial professional use you should be willing to go through a period of learning and practice to achieve facility in using the tool. Just as one likes programming language editors that provide skeletons, statement completion, and incremental syntax and style checking, these features can be useful in editors for design notations.

During the creation of software specifications and designs, the tool features that are readily apparent include ease of initial capture, including catalogs of reusable units or graphics, "automatic expansion or completion" of parts of the description, checks for correct use of the notation and conformance to any style guide, type checking, and incremental checking for internal consistency. Some tools also offer features such as automatic calculation of preconditions, design measurements, simulation, capture of traceability information, and theorem proving.

Tools that provide checking and verification features that can be used during creation—that is, ones not requiring a complete description of a design—are useful for two reasons. The feedback provided can help you avoid or remove mistakes, and if you describe something in a way that results in the tool not being able to provide its calculations or analysis automatically, you know you need to state the description in a different, usually simpler, way.

While quite useful for security concerns during specification of behavior and security constraints, and design, the features mentioned so far are already available in general design tools and many are oriented toward showing correctness or compliance. In addition, many modeling approaches and tools originally created for safety can be useful in security as well. [Alexander 2005] and [Despotou 2007] are recent examples with promise, and [Stavridou 1998] provides an insightful if somewhat technical comparison of safety and security that also includes a substantial discussion of security modeling.

In contrast to showing correctness or compliance with constraints, recently tools especially made for security more frequently are oriented towards discovering particular problems, for example through the use of attack trees or patterns. Microsoft's Threat Modeling Tool has already been mentioned. Examples elsewhere of tools with similar objectives include several code-oriented tools and runtime tools, including testing tools, and others such as the tool in [Liang 2005] that attempts to recognize new attack patterns during operation.

Putting all these considerations together to make selections can be difficult. Few tool selection efforts have appeared in the literature, and, as always, one should beware of simple-minded decision-making processes such as counting the numbers of features. Two published efforts appear suggestive of how you might approach the problem for security tools. Andrew J. Kornecki has performed a series of studies of tools for high-assurance software for the U.S. Federal Aviation Administration.6 His work has been driven in part by the requirements of DO 178B, the aviation safety standard. An example is [Kornecki 2005]. The second is a report resulting from a Praxis High-Integrity Systems study for the UK government on code analysis tools as part of an investigation on improving the evaluation at the Common Criteria's EAL4 [Jackson 2004].

In the end, you need to understand your situation, problems and objectives, system, software process, and the possibilities for tool use, as well as your readiness to use various tools. Avoid tools with aspects that might be "fatal" to successful adoption and use in your organization, such as mathematical notation or logic that people might not understand or be willing to use. You also should consider others' experience with the tool and the possibilities of pilot projects to evaluate tools, as well as the introduction of different capabilities or tools over some length of time. While a highly structured decision process may not be necessary, you need to take all these characteristics and situational factors into account and attempt to select an affordable set of tools with the best cost-benefit-time-quality tradeoff for you.

SUMMARY

This article has walked at a summary level through most of the important issues and areas of knowledge concerning security-relevant requirements and design modeling tools. Additional concepts, explanations, examples, details, and references regarding tools are provided in the other article(s) in this Modeling Tools content area, and related material appears in other BSI areas including Policy, Assurance Cases, Code Analysis Tools, and Architecture. Finally, you should realize that despite anything said above, modeling and modeling tools are not magic, and they seldom replace the human design skills, technology and domain knowledge, cleverness, hard thinking, and persistence required to create good designs. However, if you possess these, the right tools can move you closer to appearing to be a magician.

This material is based upon work funded and supported by Department of Homeland Security under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center sponsored by the United States Department of Defense.

DM-0001120