Threat Risk Modeling

From OWASP

When you start a web application design, it is essential to apply threat modeling; otherwise you will squander resources, time, and money on useless controls that fail to focus on the real threats. There are multiple approaches to threat modeling, as listed below:

Software centric threat modeling
Security centric threat modeling
Asset or risk centric threat modeling. Below represents a mixture of Threat Modeling tools and industry references. The method used to assess risk is not nearly as important as actually performing a structured threat risk modeling. Microsoft notes that the single most important factor in their security improvement program was the corporate adoption of threat risk modeling. One of many considerations is Microsoft's threat modeling process. It is simple to adopt by designers, developers, code reviewers, and the quality assurance team. The following sections provide some overview information (or see Section 6.9, Further Reading, for additional resources).

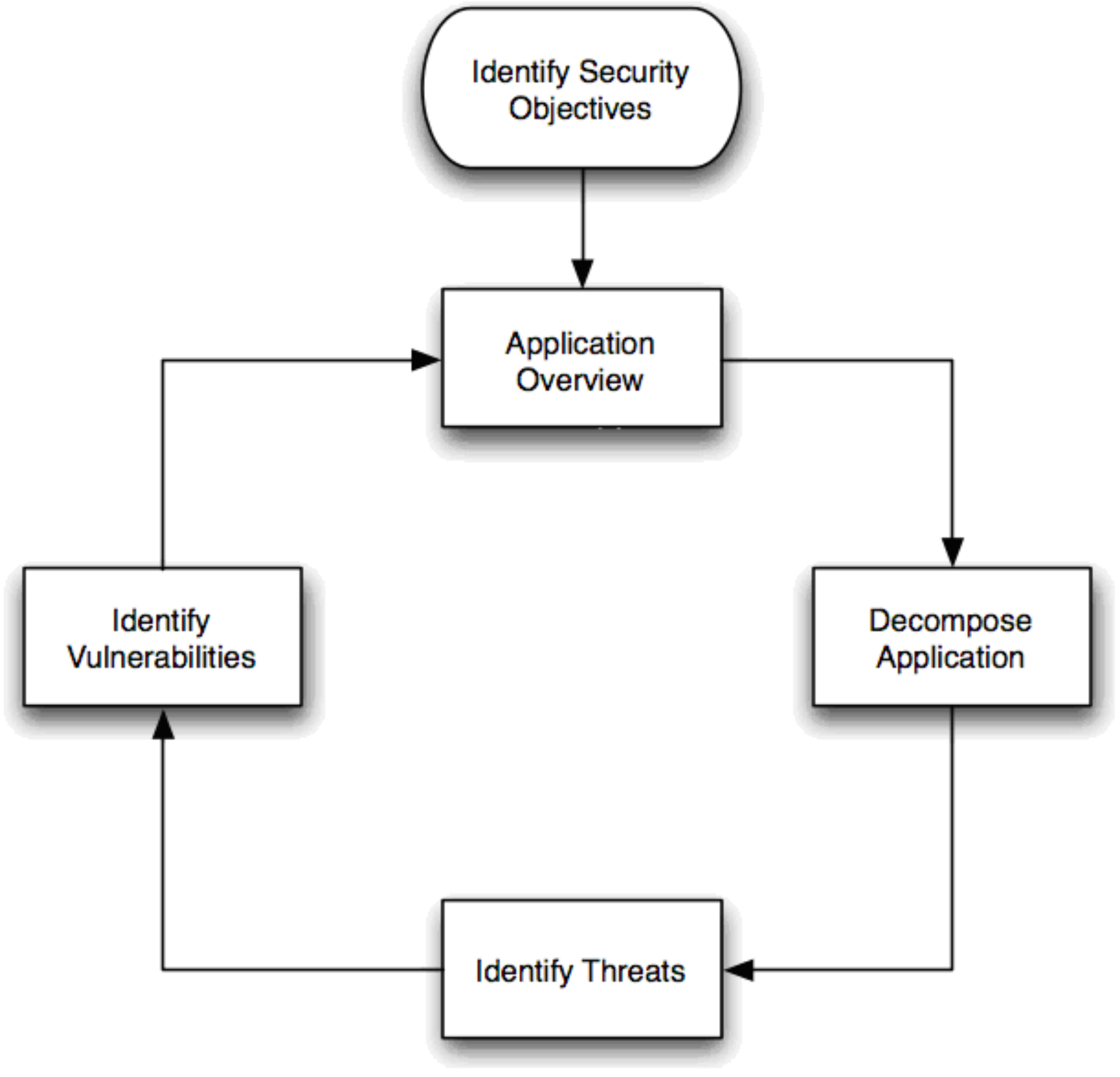
Threat Risk Modeling

Threat risk modeling is an essential process for secure web application development. It allows organizations to determine the correct controls and to produce effective countermeasures within budget. For example, there is little point in spending $100,000 for fraud control for a system that has negligible fraud risk.

== Performing threat risk modeling using the Microsoft Threat Modeling Process == //OLD - need to replace this as these steps do not factor in Impact or probabilistic threat scenarios. R=Tp * Vp * I

The threat risk modeling process has five steps, enumerated below and shown graphically in Figure 1. They are:

1. Identify Security Objectives
2. Survey the Application
3. Decompose it
4. Identify Threats
5. Identify Vulnerabilities



Let’s consider the steps in more detail.

Identify Security Objectives

The business (or project management) leadership, in concert with the software development and quality assurance teams, all need to understand the security objectives. To facilitate this, start by breaking down the application’s security objectives into the following categories:

- **Identity:** Does the application protect user identity from abuse? Are there adequate controls in place to ensure evidence of identity (as required for many banking applications?)
- **Financial:** Assess the level of risk the organization is prepared to absorb in remediation, as a potential financial loss. For example, forum software may have a lower estimated financial risk than an Internet banking application.
- **Reputation:** Quantify or estimate of the loss of reputation derived from the application being misused or successfully attacked.
- **Privacy and Regulatory:** To what extent will the application have to protect user data? Forum software by its nature is public, but a tax preparation application is subject to tax regulations and privacy legislation requirements in most countries.
- **Availability Guarantees:** Is the application required to be available per a ***Service Level Agreement (SLA)*** or similar guarantee? Is it a nationally protected infrastructure? To what level will the application have to be available? High availability techniques are significantly more expensive, so applying the correct controls up front will save a great deal of time, resources, and money.

This is by no means an exhaustive list, but it gives an idea of some of the business risk decisions leading into selecting and building security controls.

Other sources of risk guidance come from:

- Laws (such as privacy or finance laws)
- Regulations (such as banking or e-commerce regulations)
- Standards (such as ISO 17799)
- Legal Agreements (such as payment card industry standards or merchant agreements)
- Corporate Information Security Policy

Application Overview

Once the security objectives have been defined, analyze the application design to identify the ***components***, ***data flows***, and ***trust boundaries***.

Do this by surveying the application's architecture and design documentation. In particular, look for UML component diagrams. Such high level component diagrams are generally sufficient to understand how and why data flows to various places. For example, data movement across a trust boundary (such as from the Internet to the web tier, or from the business logic to the database server), needs to be carefully analyzed, whereas data that flows within the same trust level does not need as much scrutiny.

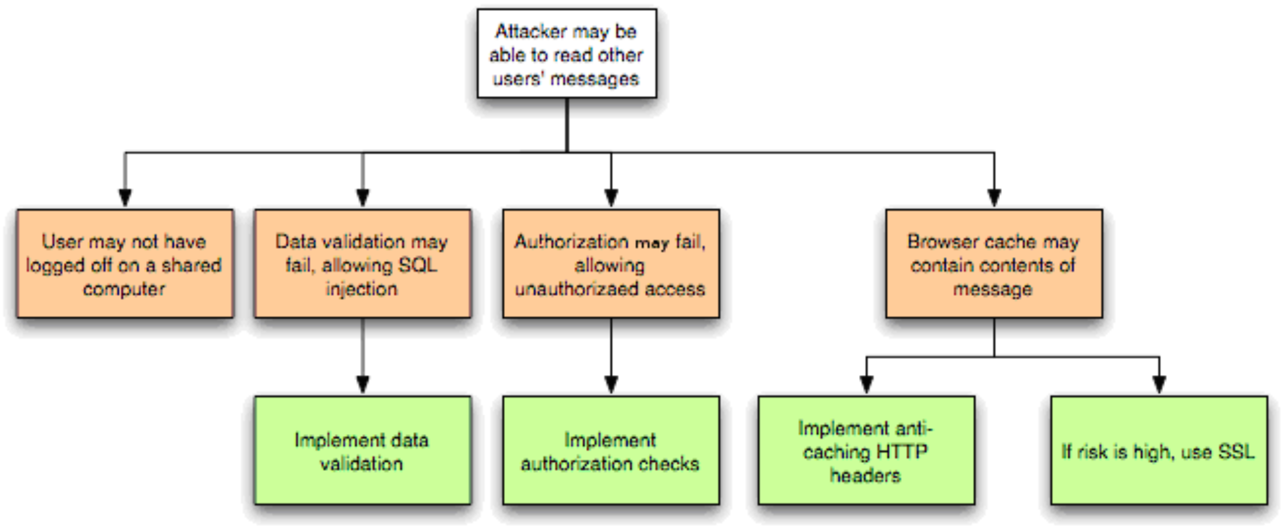
Decompose Application

Once the application architecture is understood then decompose it further, to identify the features and modules with a security impact that need to be evaluated. For example, when investigating the authentication module, it is necessary to understand how data enters the module, how the module validates and processes the data, where the data flows, how the data is stored, and what fundamental decisions and assumptions are made by the module.

Identify Threats

It is impossible to write down unknown threats, but it is likewise unlikely that new malware will be created to exploit new vulnerabilities within custom systems. Therefore, concentrate on known risks, which can be easily demonstrated using tools or techniques from Bugtraq.

Microsoft suggests two different approaches for writing up threats. One is a threat graph, as shown in Figure 2, and the other is a structured list.


Attacker may be able to read other users' messages
User may not have logged off on a shared computer
Data validation may fail, allowing SQL injection
Authorization may fail, allowing unauthorizaed access
Browser cache may contain contents of message
Implement data validation
Implement authorization checks
Implement anti-caching HTTP headers
If risk is high, use SSL

Typically, a threat graph imparts more information quickly but it takes longer to construct, while a structured list is easier to create but it will take longer for the threat impacts to become obvious.

1. Attacker may be able to read other user’s messages
2. User may not have logged off on a shared PC
3. Data validation may allow SQL injection
4. Implement data validation
5. Authorization may fail, allowing unauthorized access
6. Implement authorization checks
7. Browser cache may contain contents of message
8. Implement anti-caching directive in HTTP headers
9. If eavesdropping risk is high, use SSL

Note that it takes a motivated attacker to exploit a threat; they generally want something from your application or to obviate controls. To understand the relevant threats, use the following categories to understand who might attack the application:

- **Accidental Discovery:** An ordinary user stumbles across a functional mistake in your application, just using a web browser, and gains access to privileged information or functionality.
- **Automated Malware:** Programs or scripts, which are searching for known vulnerabilities, and then report them back to a central collection site.
- **The Curious Attacker:** a security researcher or ordinary user, who notices something wrong with the application, and decides to pursue further.
- **Script Kiddies:** Common renegades, seeking to compromise or deface applications for collateral gain, notoriety, or a political agenda, perhaps using the attack categories described in the *OWASP Web Application Penetration Checklist*.
- **The Motivated Attacker:** Potentially, a disgruntled staff member with inside knowledge or a paid professional attacker.
- **Organized Crime:** Criminals seeking high stake payouts, such as cracking e-commerce or corporate banking applications, for financial gain.

It is vital to understand the level of attacker you are defending against. For example, a motivated attacker, who understands your internal processes is often more dangerous than script kiddies.

STRIDE

STRIDE is a classification scheme for characterizing known threats according to the kinds of exploit that are used (or motivation of the attacker). The STRIDE acronym is formed from the first letter of each of the following categories.

Spoofing Identity "Identity spoofing" is a key risk for applications that have many users but provide a single execution context at the application and database level. In particular, users should not be able to become any other user or assume the attributes of another user.

Tampering with Data Users can potentially change data delivered to them, return it, and thereby potentially manipulate client-side validation, GET and POST results, cookies, HTTP headers, and so forth. The application should not send data to the user, such as interest rates or periods, which are obtainable only from within the application itself. The application should also carefully check data received from the user and validate that it is sane and applicable before storing or using it.

Repudiation Users may dispute transactions if there is insufficient auditing or recordkeeping of their activity. For example, if a user says, "But I didn't transfer any money to this external account!", and you cannot track his/her activities through the application, then it is extremely likely that the transaction will have to be written off as a loss.

Therefore, consider if the application requires non-repudiation controls, such as web access logs, audit trails at each tier, or the same user context from top to bottom. Preferably, the application should run with the user's privileges, not more, but this may not be possible with many off-the-shelf application frameworks.

Information Disclosure Users are rightfully wary of submitting private details to a system. If it is possible for an attacker to publicly reveal user data at large, whether anonymously or as an authorized user, there will be an immediate loss of confidence and a substantial period of reputation loss. Therefore, applications must include strong controls to prevent user ID tampering and abuse, particularly if they use a single context to run the entire application.

Also, consider if the user's web browser may leak information. Some web browsers may ignore the no caching directives in HTTP headers or handle them incorrectly. In a corresponding fashion, every secure application has a responsibility to minimize the amount of information stored by the web browser, just in case it leaks or leaves information behind, which can be used by an attacker to learn details about the application, the user, or to potentially become that user.

Finally, in implementing persistent values, keep in mind that the use of hidden fields is insecure by nature. Such storage should not be relied on to secure sensitive information or to provide adequate personal privacy safeguards.

Denial of Service Application designers should be aware that their applications may be subject to a denial of service attack. Therefore, the use of expensive resources such as large files, complex calculations, heavy-duty searches, or long queries should be reserved for authenticated and authorized users, and not available to anonymous users.

For applications that do not have this luxury, every facet of the application should be engineered to perform as little work as possible, to use fast and few database queries, to avoid exposing large files or unique links per user, in order to prevent simple denial of service attacks.

Elevation of Privilege If an application provides distinct user and administrative roles, then it is vital to ensure that the user cannot elevate his/her role to a higher privilege one. In particular, simply not displaying privileged role links is insufficient. Instead, all actions should be gated through an authorization matrix, to ensure that only the permitted roles can access privileged functionality.

DREAD

DREAD is a classification scheme for quantifying, comparing and prioritizing the amount of risk presented by each evaluated threat. The DREAD acronym is formed from the first letter of each category below.

DREAD modeling influences the thinking behind setting the risk rating, and is also used directly to sort the risks. The DREAD algorithm, shown below, is used to compute a risk value, which is an average of all five categories.

Risk_DREAD = (DAMAGE + REPRODUCIBILITY + EXPLOITABILITY + AFFECTED USERS + DISCOVERABILITY) / 5

The calculation always produces a number between 0 and 10; the higher the number, the more serious the risk.

Here are some examples of how to quantify the DREAD categories.

Damage Potential

- If a threat exploit occurs, how much damage will be caused?
  - 0 = Nothing
  - 5 = Individual user data is compromised or affected.
  - 10 = Complete system or data destruction

Reproducibility

- How easy is it to reproduce the threat exploit?
  - 0 = Very hard or impossible, even for administrators of the application.
  - 5 = One or two steps required, may need to be an authorized user.
  - 10 = Just a web browser and the address bar is sufficient, without authentication.

Exploitability

- What is needed to exploit this threat?
  - 0 = Advanced programming and networking knowledge, with custom or advanced attack tools.
  - 5 = Malware exists on the Internet, or an exploit is easily performed, using available attack tools.
  - 10 = Just a web browser

Affected Users

- How many users will be affected?

    - 0 = None
    - 5 = Some users, but not all
    - 10 = All users

Discoverability

- How easy is it to discover this threat?
    - 0 = Very hard to impossible; requires source code or administrative access.
    - 5 = Can figure it out by guessing or by monitoring network traces.
    - 9 = Details of faults like this are already in the public domain and can be easily discovered using a search engine.
    - 10 = The information is visible in the web browser address bar or in a form.

Note: When performing a security review of an existing application, "Discoverability" will often be set to 10 by convention, as it is assumed the threat issues will be discovered.

Note: Using DREAD can be difficult at first. It may be helpful to think of Damage Potential and Affected Users in terms of Impact, while thinking of Reproducibility, Exploitability, and Discoverability in terms of Probability. Using the Impact vs Probability approach (which follows best practices such as defined in NIST-800-30), I would alter the formula to make the Impact score equal to the Probability score. Otherwise the probability scores have more weight in the total.

Alternative Threat Modeling Systems

OWASP recognizes that the adoption of the Microsoft modeling process may not fit all organizations. If STRIDE and DREAD are unacceptable for some reason, we recommend that your organization "dry run" the other threat risk models discussed against an existing application or design. This will allow you to determine which approach works best for you, and to adopt the most appropriate threat modeling tools for your organization.

In summary, performing threat modeling provides a far greater return than most any other control in this Guide. Therefore, make threat risk modeling an early priority in your application design process.

Trike

Trike is a threat modeling framework with similarities to the Microsoft threat modeling processes. However, Trike differs because it uses a risk based approach with distinct implementation, threat, and risk models, instead of using the STRIDE/DREAD aggregated threat model (attacks, threats, and weaknesses). From the Trike paper, Trike's goals are:

- With assistance from the system stakeholders, to ensure that the risk this system entails to each asset is acceptable to all stakeholders.
- Be able to tell whether we have done this.
- Communicate what we've done and its effects to the stakeholders.
- Empower stakeholders to understand and reduce the risks to them and other stakeholders implied by their actions within their domains.

For more information on Trike, please see Section 6.9, reference 8.

AS/NZS 4360:2004 Risk Management

The Australian/New Zealand Standard AS/NZS 4360, first issued in 1999, and revised in 2004, is the world's first formal standard for documenting and managing risk and is still one of the few formal standards for managing it. The standard's approach is simple (it's only 28 pages long), flexible, and iterative. Furthermore, it does not lock organizations into a particular risk management methodology, provided the methodology fulfils the AS/NZS 4360 five steps. It also provides several sets of risk tables as examples, and allows organizations to freely develop and adopt their own.

The five steps of the AS/NZS 4360 process are:

- **Establish Context:** Establish the risk domain, i.e., which assets/systems are important?
- **Identify the Risks:** Within the risk domain, what specific risks are apparent?
- **Analyze the Risks:** Look at the risks and determine if there are any supporting controls in place.
- **Evaluate the Risks:** Determine the residual risk.
- **Treat the Risks:** Describe the method to treat the risks so that risks selected by the business will be mitigated.

AS/NZS 4360 assumes that risk will be managed by an ***operational risk group***, and that the organization has adequate skills and risk management resources in house to identify, analyze, and treat the risks.

The advantages of AS/NZS 4360:

- AS/NZS 4360 works well as a risk management methodology for organizations requiring Sarbanes-Oxley compliance.
- AS/NZS 4360 works well for organizations that prefer to manage risks in a traditional way, such as just using likelihood and consequence to determine an overall risk.
- AS/NZS 4360 is familiar to most risk managers worldwide, and your organization may already have implemented an AS/NZS 4360 compatible approach.
- You are an Australian organization, and may be required to use it if you are audited on a regular basis, or to justify why you aren't using it. Luckily, the STRIDE/DREAD model discussed earlier is AS/NZS 4360 compatible.

The limitations of AS/NZS 4360:

- The AS/NZS 4360 approach works best for business or systemic risks than for technical risks.
- AS/NZS 4360 does not define the methodology to perform a structured threat risk modeling exercise.
- As AS/NZS 4360 is a generic framework for managing risk, it does not provide any structured method to enumerate web application security risks.

Although AS/NZS 4360 may be used to rank risks for security reviews, the lack of structured methods of enumerating threats for web applications makes it less desirable than other methodologies described earlier.

CVSS

The US Department of Homeland Security (DHS) established the NIAC Vulnerability Disclosure Working Group, which incorporates input from Cisco Systems, Symantec, ISS, Qualys, Microsoft, CERT/CC, and eBay. One of the group's outputs is the ***Common Vulnerability Scoring System (CVSS).***

The advantages of CVSS:

- You have just received notification from a security researcher or other source that your product has vulnerability, and you wish to ensure that it has an accurate and normalized severity rating, so as to alert your customers to the appropriate level of action required when you release the patch.
- You are a security researcher, and have found several threat exploits within an application. You would like to use the CVSS ranking system to produce reliable risk rankings, to ensure that the ISV will take the exploits seriously as indicated by their rating.
- CVSS has been recommended by the working group for use by US Government departments. However, it is unclear if it will become policy or be widely adopted at the time of this writing.

The limitations of CVSS:

- CVSS does not find or reduce the attack surface area (i.e. design flaws), or help enumerate risks within any arbitrary piece of code, as it is just a scoring system, not a modeling methodology.
- CVSS is more complex than STRIDE/DREAD, as it aims to calculate the risk of announced vulnerabilities as applied to deployed software and environmental factors.
- The CVSS risk ranking is complex – a spreadsheet is required to calculate the risk components as the assumption behind CVSS is that a specific vulnerability has been identified and announced, or a worm or Trojan has been released targeting a small number of attack vectors.
- The overhead of calculating the CVSS risk ranking is quite high if applied to a thorough code review, which may have 250 or more threats to rank.

OCTAVE

OCTAVE is a heavyweight risk methodology approach originating from Carnegie Mellon University's Software Engineering Institute (SEI) in collaboration with CERT. OCTAVE focuses on organizational risk, not technical risk. OCTAVE comes in two versions: Full OCTAVE, for large organizations, and OCTAVE-S for small organizations, both of which have specific catalogs of practices, profiles, and worksheets to document the modeling outcomes.

OCTAVE is popular with many sites and is useful when:

- Implementing an organizational culture of risk management and controls becomes necessary.
- Documenting and measuring business risk becomes timely.
- Documenting and measuring the overall IT security risk, particularly as it relates to the corporate IT risk management, becomes necessary.
- When documenting risks surrounding complete systems becomes necessary.
- To accommodate a fundamental reorganization, such as when an organization does not have a working risk methodology in place, and requires a robust risk management framework to be put in place.

The limitations of OCTAVE are:

- OCTAVE is incompatible with AS/NZS 4360, as it mandates Likelihood = 1 (i.e., It assumes a threat will always occur) and this is inappropriate for many organizations. OCTAVE-S makes the inclusion of this probability optional, but this is not part of the more comprehensive OCTAVE standard.
- Consisting of 18 volumes, OCTAVE is large and complex, with many worksheets and practices to implement.

- It does not provide a list of “out of the box” practices for assessing and mitigating web application security risks.

Because of these issues, OWASP does not anticipate that OCTAVE will be used at large by application designers or developers, because it fails to take threat risk modeling into consideration, which is useful during all stages of development, by all participants, to reduce the overall risk of an application becoming vulnerable to attack.

ThreatModel SDK

The ThreatModel SDK is a minimalistic Java library that provides a basic vendor-neutral object model along with the ability to parse reports generated from common threat modeling tools. Supported Threat Modeling Tools:

- Microsoft Threat Modeling Tool 2016

Planned Threat Modeling Tools:

- Mozilla SeaSponge

For more information visit: https://github.com/stevespringett/threatmodel-sdk

Conclusion

In this chapter, we have touched on the basic principles of threat risk modeling, risk management, and web application security. Applications that leverage the underlying intent of these principles will be more secure than their counterparts, which will only be minimally compliant just by including specific controls.

Further Reading

- Threat Analysis & Modeling v2.1.2 (http://www.microsoft.com/downloads/details.aspx?FamilyId=59888078-9DAF-4E96-B7D1-944703479451), © Microsoft Corporation, 2007.
- Threat Modeling Web Applications (http://msdn.microsoft.com/library/ms978516.aspx), J.D. Meier, Alex Mackman, Blaine Wastell, © Microsoft Corporation, May 2005.
- Improving Web Application Security: Threats and Countermeasures (http://msdn.microsoft.com/library/ms994921.aspx), J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan, © Microsoft Corporation, June 2003.
- Threat Modeling (http://www.microsoft.com/downloads/details.aspx?FamilyID=62830f95-0e61-4f87-88a6-e7c663444ac1&displaylang=en), Frank Swiderski and Window Snyder, Microsoft Press, June 2004, ISBN 0-7356-1991-3.
- Writing Secure Code, 2nd Edition, Howard and LeBlanc, (pp. 69 – 124), Microsoft Press, 2003, ISBN 0-7356-1722-8.
- The STRIDE Threat Model (http://msdn.microsoft.com/library/ms954176.aspx), © Microsoft Corporation, 2005.
- DREADful (http://blogs.msdn.com/david_leblanc/archive/2007/08/13/dreadful.aspx) - the DREAD system, © Microsoft Corporation, 2005.
- A Conceptual Model for Threat Modeling Applications (http://dymaxion.org/trike

/Trike_v1_Methodology_Document-draft.pdf), Saitta, Larcom, and Michael Eddington, July 2005, http://dymaxion.org/trike/.

- AS/NZS 4360:2004 Risk Management (http://www.standards.co.nz/web-shop/?action=viewSearchProduct&mod=catalog&pid=4360:2004(AS%7CNZS)), Standards Australia and Standards New Zealand.
- CVSS (http://www.dhs.gov/interweb/assetlibrary/NIAC_CyberVulnerabilitiesPaper_Feb05.pdf), U.S. Department of Homeland Security library, February 2005.
- OCTAVE (http://www.cert.org/octave/), CERT library.

Appendix: Alternative open-source Risk Management tools

- OSMR (http://sourceforge.net/projects/osmr/)
- MARCO (http://sourceforge.net/projects/marco/)
- CORAS Risk Assessment Platform (http://sourceforge.net/projects/coras/)
- ISO 17799 Risk Assessment Toolkit (http://sourceforge.net/projects/ratiso17799/)
- Easy Threat Risk Assessment (http://sourceforge.net/projects/easy-tra/)
- ARMS (http://sourceforge.net/projects/arms-17799/)
- Minaccia (http://sourceforge.net/projects/minaccia/)
- ThreatMind (http://sourceforge.net/projects/threatmind/)
- Open Source Requirements Management Tool (http://sourceforge.net/projects/osrmt/)

Reference

Development Guide Table of Contents
Retrieved from "http://www.owasp.org/index.php?title=Threat_Risk_Modeling&oldid=225484"
Categories: FIXME | OWASP Guide Project | Activity | Externally Linked Page | Threat Modeling | SAMM-TA-1
