Threat Risk Modeling

When designing your application, it is essential you design using threat risk assessed controls, otherwise you will squander resources, time and money on useless controls and not enough on the real risks.

The method you use to determine risk is not nearly as important as actually performing structured threat risk modeling. Microsoft notes in REFERENCE that the single most important improvement in their security improvement program was the universal adoption of threat modeling.

OWASP has chosen Microsoft’s threat modeling process as it works well for the unique challenges facing application security, and is simple to learn and adopt by designers, developers, and code reviewers.

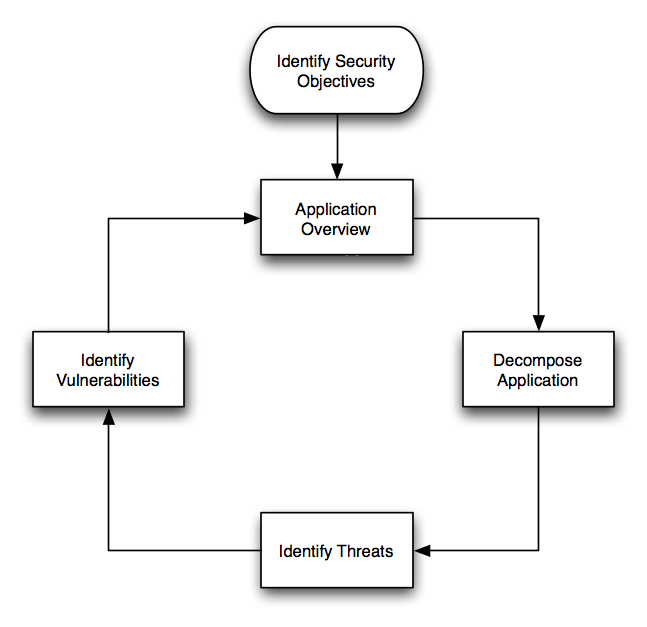
Threat Risk Modeling using the Microsoft Threat Modeling Process

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

Performing threat risk modeling

There are five steps in the threat modeling process. Microsoft provides a threat modeling tool written in .NET to assist with tracking and displaying threat trees. You may find using this tool helpful for larger or long-lived projects.

Threat Model Flow



Identify Security Objectives

The business (or organization’s leadership) in concert with the development team needs to understand the likely security objectives. The application’s security objectives need to be broken down into:

* Identity: does this application protect user’s identity from misuse? Are there adequate controls to ensure evidence of identity (required for many banking applications)?
* Reputation: the loss of reputation derived from the application being misused or successfully attacked
* Financial: the level of risk the organization is prepared to stake in remediation potential financial loss. Forum software would have a lower financial risk than corporate Internet banking
* Privacy and regulatory: to what extent shall applications protect user’s data. Forum software is by its nature public, but a tax program is inherently bound up in tax regulation and privacy legislation in most countries
* Availability guarantees: is this software required to be available by SLA or similar agreement? Is it nationally protected infrastructure? To what level will the application need to be available? Highly available applications and techniques are extraordinarily expensive, so setting the correct controls here can 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 that lead into building technical 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 merchant agreements
* Information Security Policy

Application Overview

Once the security objectives have been defined, the application should be analyzed to determine:

* Components
* Data flows
* Trust Boundaries

The best way to do this is to obtain the application’s architecture and design documentation. Look for UML component diagrams. The high level component diagrams are generally all that’s required to understand how and why data flows to various places. Data which crosses a trust boundary (such as from the Internet to the front end code or from business logic to the database server), needs to be carefully analyzed, whereas data which flows within the same trust level does not need as much scrutiny.

Decompose application

Once the application architecture is understood, the application needs to be decomposed, which is to say that features and modules which have a security impact need to be decomposed. For example, when investigating the authentication module, it is necessary to understand how data enters the authentication module, how the module validates and processes the data, where the data flows, if data is stored, and what decisions are made by the module.

Document the known threats

It is impossible to write down unknown threats, and it is unlikely for many custom systems that new malware will be created to deal with new vulnerabilities. Instead, concentrate on risks which are known, which can easily be demonstrated using tools or from Bugtraq.

When writing up a threat, Microsoft suggests two different approaches. One is a threat graph, and the other is just a structured list. Typically, a threat graph imparts a lot more information in a shorter time for the reader but takes longer to construct, and a structured list is much easier to write but takes longer for the impact of the threats to become obvious.

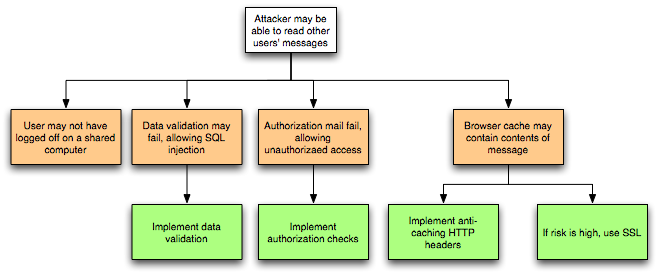


Figure : Threat Tree

Attacker may be able to read other user’s messages

User may not have logged off on a shared PC

Data validation may allow SQL injection

Implement data validation

Authorization may fail, allowing unauthorized access

Implement authorization checks

Browser cache may contain contents of message

Implement anti-caching HTTP headers

If risk is high, use SSL

Threats are motivated attackers; they generally want something from your application or obviate controls. To understand what threats are applicable, use the security objectives to understand who might attack the application:

* Accidental discovery: authorized users stumble across a mistake in your application logic using just a browser
* Automated malware (searching for known vulnerabilities but with little malice or smarts)
* Curious Attacker (such as security researchers or users who notice something wrong with your application and test further)
* Script kiddie: computer criminals attacking or defacing applications for “respect” or political motives – using techniques described here and in the OWASP Testing Guides to compromise your application
* Motivated attacker (such as disgruntled staff or paid attacker)
* Organized crime (generally for higher risk applications, such as e-commerce or banking)

It is vital to understand the level of attacker you are defending against. An informed attacker who understands your processes is far more dangerous than a script kiddie, for example.

STRIDE

Spoofing identity

Spoofing identity is a key risk of applications that have many users but use a single execution context at the application and database level. Users must not be able to act as any other user, or become that user.

Tampering with data

Users can change any data delivered to them, and can thus can change client-side validation, GET and POST data, cookies, HTTP headers, and so on. The application should not send data to the user, such as interest rates or periods that are obtainable within the application itself. The application must carefully check any data received from the user to identify if it is sane and applicable.

Repudiation

Users can dispute transactions if there is insufficient traceability and auditing of user activity. For example, if a user says “I didn’t transfer money to this external account”, and you cannot track their activities from front to back of the application, it is extremely likely that the transaction will have to be written off.

Applications should have adequate repudiation controls, such as web access logs, audit trails at every tier, and a user context from top to bottom. Preferably, the application should run as the user, but this is often not possible with many frameworks.

Information Disclosure

Users are wary of submitting private details to a system. If it is possible for an attacker to reveal user details, whether anonymously or as an authorized user, there will be a period of reputation loss. Applications must include strong controls to prevent user ID tampering, particularly when they use a single account to run the entire application.

The user’s browser can leak information. Not every browser correctly implements the no caching policies requested by the HTTP headers. Every application has a responsibility to minimize the amount of information stored by a browser, just in case it leaks information and can be used by an attacker to learn more about the user or even become that user.

Denial of Service

Applications should be aware that they could be abused by a denial of service attack. For authenticated applications, expensive resources such as large files, complex calculations, heavy-duty searches, or long queries should be reserved for authorized users, not anonymous users.

For applications which do not have this luxury, every facet of the application must be implemented to perform as little work as possible, use fast (or no) database queries, and not expose large files, or providing unique links per user to prevent a simple denial of service attack.

Elevation of Privilege

If an application provides user and administration roles, it is vital to ensure that the user cannot elevate themselves to any higher privilege roles. In particular, simply not providing the user links is insufficient – all actions must be gated through an authorization matrix to ensure that the only the right roles can access privileged functionality.

DREAD

DREAD is used to form part of the thinking behind the risk rating, and is used directly to sort the risks.

DREAD is used to compute a risk value, which is an average of all five elements:

RiskDREAD = ( DAMAGE + REPRODUCABILITY + EXPLOITABILITY + AFFECTED USERS + DISCOVERABILITY ) / 5

This produces a number between 0 and 10. The higher the number, the more serious the risk.

Damage Potential

If a threat is realized, how much damage is caused?

|  |  |  |
| --- | --- | --- |
| 0 = nothing | 5 = individual user data is compromised or affected | 10 = complete system destruction |

Reproducibility

How easy is it to reproduce this threat?

|  |  |  |
| --- | --- | --- |
| 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 = requires just a browser and the address bar without being logged on |

Exploitability

What do you need to have to exploit this threat?

|  |  |  |
| --- | --- | --- |
| 0 = advanced programming and networking skills, advanced or custom attack tools | 5 = malware exists, or easily performed using normal attack tools | 10 = just a browser |

Affected Users

How many users will this threat affect?

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

Discoverability

How easy is it to discover this threat? When performing a code review of an existing application, “Discoverability” is usually set to 10 as it has to be assumed that these issues will be discovered.

|  |  |  |
| --- | --- | --- |
| 0 = very hard to impossible. Requires source or system access | 5 = Could figure it out from guessing or watching network traces | 9 = Details of faults like this are in the public domain, and can be discovered using Google  10 = It’s in the address bar or in a form |

Further Reading on Threat Modeling

AS/NZS 4360:2004 Risk Management, available from Standards Australia and Standards New Zealand:

<http://shop.standards.co.nz/productdetail.jsp?sku=4360%3A2004%28AS%2FNZS%29>

OCTAVE:

Microsoft has several well-regarded texts on threat modeling. These include:

Howard and LeBlanc, Writing Secure Code, 2nd Edition, pp 69 – 124, © 2003 Microsoft Press, ISBN 0-7356-1722-8

Meier et al, Improving Web Application Security: Threats and Countermeasures, © 2003 Microsoft Press

Alternative Threat Modeling Systems

OWASP recognizes that the adoption of a Microsoft modeling process may be a controversial choice in some organizations. If STRIDE / DREAD is unacceptable due to unfounded prejudice, we recommend that every organization trial the various threat models against an existing application or design. This will allow the organization to determine which approach works best for them, and adopt the most appropriate threat modeling tools for their organization.

Performing threat modeling provides a far greater return than any control in this Guide. It is vital that threat modeling takes place.

Trike

Trike is a threat modeling framework with similarities to the Microsoft threat model process. However, Trike is differentiated by using a risk based approach with distinct implementation, threat and risk models, instead of using a mixed threat model (attacks, threats, and weaknesses) as represented by STRIDE / DREAD.

From the Trike paper, Trike aims to:

With assistance from the system stakeholders, 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 themselves and other stakeholders implied by their actions within their domains.

For more information, please review the “Further Reading” section below. The OWASP Guide 2.1 (due November 2005) will have more details on Trike.

AS/NZS 4360:2004 Risk Management

Australian Standard / New Zealand Standard AS/NZS 4360, first issued in 1999, is the world’s first formal standard for documenting and managing risk, and is still one of the few formal standards for managing risk. It was updated in 2004.

AS/NZS 4360’s approach is simple (it’s only 28 pages long) and flexible, and does not lock organizations into any particular method of risk management as long as the risk management fulfils the AS/NZS 4360 five steps. It provides several sets of risk tables and allows organizations to adopt their own.

The five major components of AS/NZS 4360’s iterative approach are:

Establish the context – establish what is to be risk treated, i.e. which assets / systems are important

Identify the risks – within the systems to be treated, what risks are apparent?

Analyze the risks – look at the risks and determine if there are any supporting controls

Evaluate the risks – determine the residual risk

Treat the risks – describe the method to treat the risks so that risks selected by the business can be mitigated.

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

Why you would use 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 using just 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 4360 compatible approach

You are an Australian organization, and thus may be required to use it if you are audited externally on a regular basis, or justify why you are not using it. Luckily, the STRIDE / DREAD model referred to above is AS/NZS 4360 compatible.

Why you would not use AS/NZS 4360:

AS/NZS 4360’s approach works far better for business or systemic risks than technical risks

AS/NZS 4360 does not discuss methods 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 4360 can 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 methods.

CVSS

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

Why you would use CVSS:

You have just received notification from a security researcher or other source that your product has a vulnerability, and you wish to ensure that it has a trustworthy severity rating so as to alert your users to the appropriate level of action required when you release the patch

You are a security researcher, and have found several exploits for a program. 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 compared to their rating.

The use of CVSS is recommended for use by US government departments by the working group – it is unclear if this is policy at the time of writing.

Why you would not use CVSS:

CVSS does not find or reduce attack surface area (i.e. design flaws), nor help enumerate possible risks from any arbitrary piece of code as it is not designed for that purpose.

CVSS is more complex than STRIDE / DREAD, as it aims to model the risk of announced vulnerabilities as applied to released software.

CVSS risk ranking is complex – a spreadsheet is required to calculate the risks as the assumption behind CVSS is that a single risk has been 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 from CMU’s Software Engineering Institute in collaboration with CERT. OCTAVE is targeted not at technical risk, but organizational risk.

OCTAVE consists of two versions: OCTAVE – for large organizations and OCTAVE-S for small organizations, both of which have catalogs of practices, profiles, and worksheets to document the OCTAVE outcomes. OCTAVE is popular with many sites.

OCTAVE is useful when:

Implementing a culture of risk management and control within an organization

Documenting and measuring business risk

Documenting and measuring overall IT security risk, particularly as it relates to whole of company IT risk management

Documenting risks surrounding complete systems

When an organization is mature, does not have a working risk methodology in place, and requires a robust risk management framework to be put in place

The downsides of Octave are:

OCTAVE is incompatible with AS 4360, as it forces Likelihood=1 (i.e. a threat will always occur). This is also inappropriate for many organizations. OCTAVE-S has an optional inclusion of probability, but this is not part of OCTAVE.

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 web application security risks

OWASP does not expect OCTAVE to be used by designers or developers of applications, and thus it misses the raison d’être of threat modeling – which is to be used during all stages of development by all participants to reduce the risk of the application becoming vulnerable to attack.

Comparing threat modeling approaches

Here’s how roughly the CVSS measures up to STRIDE / DREAD:

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | Attribute | Description | Closest STRIDE / DREAD |
| CVSS Base Metrics | Access vector | Local or remote access? | ~ Exploitability |
| CVSS Base Metrics | Access complexity | How hard to reproduce the exploit | Reproducibility |
| CVSS Base Metrics | Authentication | Anonymous or authenticated? | ~ Exploitability |
| CVSS Base Metrics | Confidentiality impact | Impact of confidentiality breach | Information Disclosure |
| CVSS Base Metrics | Integrity impact | Impact of integrity breach | Tampering |
| CVSS Base Metrics | Availability impact | Impact of system availability breach | Denial of Service |
| CVSS Base Metrics | Impact bias | Bias equal for CIA, or biased towards one or more of CIA? | No equivalent |
| CVSS Temporal | Exploitability | How easy is the breach to exploit? | Exploitability |
| CVSS Temporal | Remediation Level | Is a fix available? | No equivalent |
| CVSS Temporal | Report confidence | How reliable is the original report of the vulnerability? | No equivalent |
| CVSS Environmental | Collateral Damage | How bad is the damage if the threat were realized? | Damage potential |
| CVSS Environmental | Target Distribution | How many servers are affected if the threat were realized? | Affected users (not directly equivalent) |

Alternatively, here is how STRIDE / DREAD map to CVSS:

| STRIDE attribute | Description | Closest CVSS attribute |
| --- | --- | --- |
| Spoofing identity | How can users obviate controls to become another user or act as another user? | No direct equivalent |
| Tampering with data | Can data be tampered with by an attacker to get the application to obviate security controls or take over the underlying systems (eg SQL injections) | Integrity |
| Repudiation | Can users reject transactions due to a lack of traceability within the application? | No direct equivalent |
| Information disclosure | Can authorization controls be obviated which would then lead to sensitive information being exposed which should not be? | Confidentiality |
| Denial of service | Can an attacker prevent authorized users from accessing the system? | Availability |
| Elevation of privilege | Can an anonymous attacker become a user, or an authenticated user act as an administrator or otherwise change to a more privileged role? | No direct equivalent |

| DREAD attribute | Description … if the threat is realized | Closest CVSS attribute |
| --- | --- | --- |
| Damage potential | What damage may occur? | Collateral damage |
| Reproducibility | How easy is it for a potential attack to work? | ~ Access complexity |
| Exploitability | What do you need (effort, expertise) to make the attack work? | Exploitability |
| Affected users | How many users would be affected by the attack? | Target distribution |
| Discoverability | How easy is for it attackers to discover the issue? | No direct equivalent |

In general, CVSS is useful for released software and the number of realized vulnerabilities is small. CVSS should produce similar risk rankings regardless of reviewer, but many of the biases built into the overall risk calculation are subjective (i.e. local and remote or which aspect of the application is more important), and thus there may disagreement of the resulting risk ranking.

STRIDE/DREAD is useful to reduce attack surface area, improve design, and eliminate vulnerabilities before they are released. It can also be used by reviewers to rank and enumerate threats in a structured way, and produce similar risk rankings regardless of reviewer.

Conclusion

In the few previous pages, we have touched on the basic principles of web application security. Applications that honor the underlying intent of these principles will be more secure than their counterparts who are minimally compliant with specific controls mentioned later in this Guide.

Further Reading

* Microsoft, *Threats and Countermeasures*, © 2003 Microsoft
* Howard and LeBlanc, *Writing Secure Code*, © 2003 Microsoft Press
* Saitta, Larcom, and Michael Eddington, *A conceptual model for threat modeling applications,* July 132005   
  <http://dymaxion.org/trike/>   
  <http://dymaxion.org/trike/Trike_v1_Methodology_Document-draft.pdf>
* CVSS  
  <http://www.dhs.gov/interweb/assetlibrary/NIAC_CyberVulnerabilitiesPaper_Feb05.pdf>