Business Threat Modeling

Risk-driven software in a post AI, post-privacy era

Update March 2026

The first version of Business threat modeling was written 20 years ago and it has aged surprisingly well. Thousands of analysts worldwide adopted the methodology in many different industries from healthcare to clinical trials to financial services to telecommunications. The companion software, PTA - Practical Threat Analysis became the inspiration for OpenCRO - the Open Risk Officer.

AI has changed the way we code. Our coding chops have atrophied. AI does the work.

But, does AI really eliminate software development? And does AI help us make safer software?

To answer these 2 questions, I divided software into 2 categories: *Human-facing* and *machine-facing.*

Human-Facing

1. Stakeholder alignment
2. Market discovery
3. Business tradeoffs
4. Regulatory strategy (in industries like life science and energy)
5. Business model validation with human buyers
6. Capital allocation

Machine-Facing

1. Code generation
2. Test generation
3. Build pipelines
4. DevOps
5. Refactoring
6. Static analysis

In personal productivity applications, human-facing activities are low cost. You make your own tradeoffs. You write specs and AI writes code.

In enterprise software and regulated systems like healthcare, the equation flips. Human interactions dominate time and cost. You cannot make tradeoffs on your own. You need to align and comply.

And when humans are involved, risk increases.

This is why the "*AI eliminates software development*" narrative is naive.

Business Threat Modeling is still relevant today.

ABSTRACT

What risks really count for your business? No question is more important for implementing an effective program of security countermeasures for your business. The management board, IT and security practitioners cannot expect to mitigate risk effectively without knowing the sources and cost of threats to the organization.

A modern organization depends on its transaction processing systems in order to conduct business. The prevailing security model predicates defense in depth of these systems. The most common strategies are to mitigate risk with network and application security products that are reactive countermeasures; blocking network ports and services, detecting known application exploits, or by blocking entry of malicious code to the network.

Are any of these security countermeasures likely to be effective in the long-term? Can attacks on a business be neutralized with defensive means only? In other words, is there a "black-box" security solution for the business? The answer is clearly no.

A reactive network defense tool such as a firewall cannot prevent exploitation of software defects and an application firewall is no replacement for in-depth understanding of company-specific source code or system configuration vulnerabilities.

Business Threat Modeling is a risk-driven software development process for software engineering teams. It employs a continuous risk analysis of complex software systems along with quantitative evaluation of how well removing software defects reduces risk.

Business Threat Modeling is based on four basic tenets that are discussed at greater length in this article. The four tenets are:

1. Security assessment of complex software systems
2. Quantitative evaluation and financial justification for security countermeasures
3. Explicit communications between developers and security
4. Sustain continuous risk reduction

THE PROBLEM: DEFECTIVE SYSTEMS ARE INSECURE SYSTEMS

This seemingly obvious observation is graphically borne out in a study that analyzed a sample of 167 customer data breaches over 20 years ago in 2005.[1] Based on data provided by the Privacy Rights Clearinghouse.[2] The study classified each event according to attack method, attacker and vulnerability exploited.

A conservative estimate showed that 49% of the events exploited software defects as shown in the below table. Theoretically we can mitigate half of the risk by removing software defects in existing applications. The question, which we will answer later, is how.

Aggregated vulnerability distribution by type		
Vulnerability type	**Total**	**Percentage**
Accidental disclosure by email	5	3.0%
Human weakness of system users/operators	13	7.8%
Unprotected computers / backup media	67	40.1%
Malicious exploits of system defects	**82**	**49.1%**
Grand Total	167	100.0%

The Carnegie Mellon Software Engineering Institute (SEI) reports that 90 percent of *all* software vulnerabilities are due to well-known defect types (for example using a hard coded server password or writing temporary work files with world read privileges). All of the SANS Top 20 Internet Security vulnerabilities are the result of "poor coding, testing and sloppy software engineering".[3]

Why don't organizations do more to improve their production software quality?

Let's examine commitment to quality at three levels in an organization: end-users, development managers and top executives.

Users are conditioned to accept unreliable software on their desktop and development managers are inclined to accept faulty software as a tradeoff to meeting a development schedule.

Executives, while committed to quality of their **own** products and services, do not find security breaches sufficient reason to become security leaders with their enterprise systems because:

a. They usually receive conflicting proposals for new information security initiatives with weak or missing financial justifications.
b. The recommended security initiatives often disrupt the business. [4]

1 2005 Breach Analysis, April 2006 http://www.software.co.il/downloads/breachAnalysis2005.xls
2 Privacy Rights Clearinghouse, http://www.privacyrights.org/
3 "Developing Secure Software, Noopur Davis, http://www.softwaretechnews.com/stn8-2/noopur.html
4 "Top-down Security", Alan Paller, http://infosecuritymag.techtarget.com/articles/1999/paller.shtml

The need to understand operational risk of software security

Network and application security products are reactive means used to **defend** the organization rather than proactive means of **understanding and reducing** operational risk.

Today's defense in depth strategy is to deploy multiple tools at the network perimeter such as firewalls, intrusion prevention and malicious content filtering. The defense-focus is primarily on **outside-in** attacks, despite the fact that the majority of attacks on customer data and intellectual property are **inside out**. The notion of trusted systems inside a hard perimeter has practically disappeared with the proliferation of Web services, SSL VPN and convergence of application transport to HTTP.

A reactive network defense tool such as a firewall cannot prevent exploitation of software defects and black-box application security that relies on checklists of vulnerabilities is no replacement for in-depth understanding of specific source code or system configuration vulnerabilities.

THE OBJECTIVE: COST EFFECTIVE SYSTEM DEFECT REDUCTION

It is rare to see systematic defect reduction projects in production software running in the enterprise, apparently, if it were easy, everyone would be doing it. So what makes it so hard?

1. Current development methodologies (including Agile) used by internal development teams are a bad fit for threat analysis of production software systems.
2. The cost of finding and fixing a bug in a production system is regarded as too high.[5]
3. The application developers and IT security teams don't usually talk to each other. The larger the organization, the more they lose when information gets lost in the cracks.

We can meet these challenges in a cost-effective way by establishing three tenets:

1. Use a risk analysis process that is suitable for production software systems.
   Collect data from all levels in the organization that touch the production system and classify defects for risk mitigation according to standard vulnerability and problem types.

2. Provide executives with financial justification for defect reduction.
   Quantify the risk in terms of assets, software vulnerabilities, and the organization's current threats.

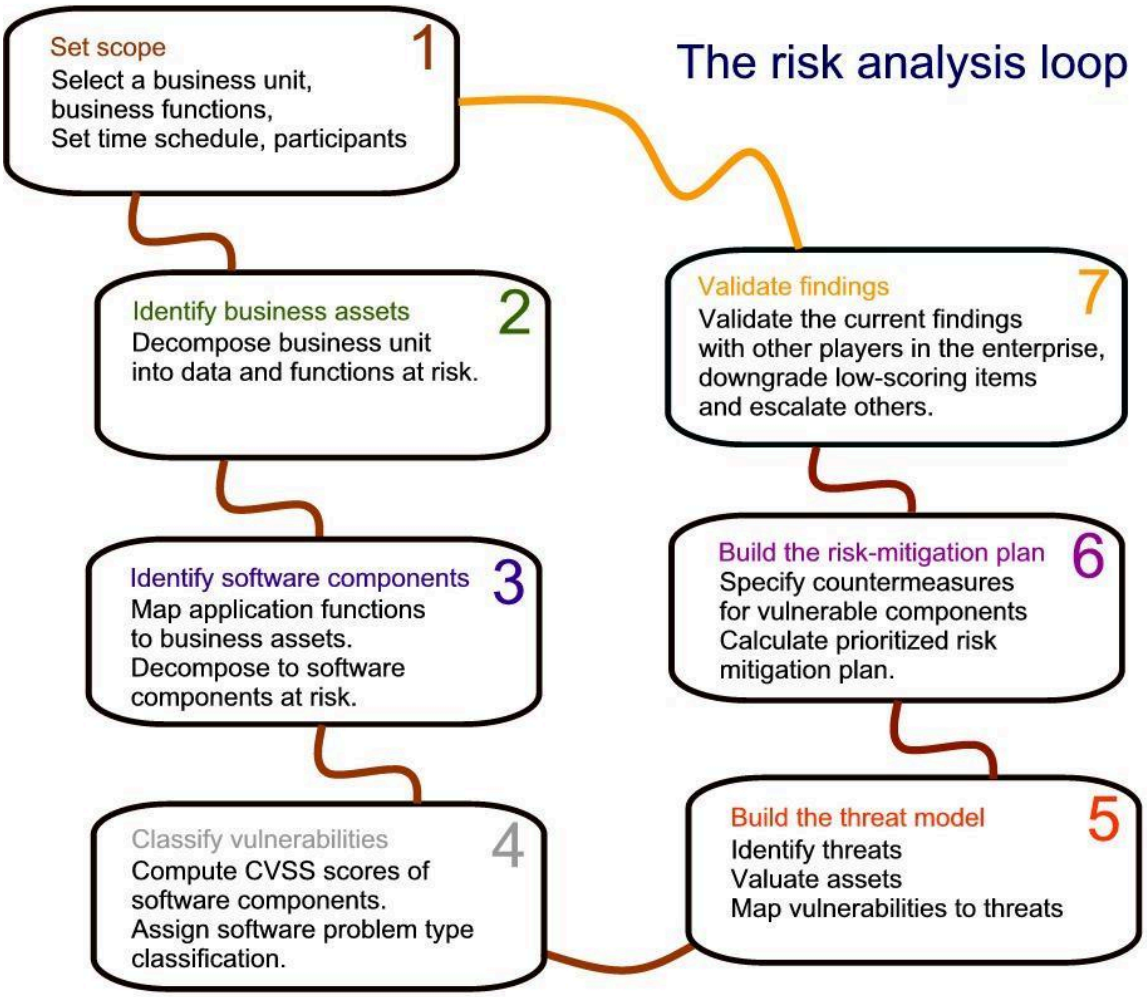
3. Require the development and IT security teams to start talking.
   Explicit communications between software developers and IT security can be facilitated by an online knowledge base and ticketing tool that provide an updated picture of well-known defects and security events.

[5] "In production, it's often 100 times more expensive than finding and fixing the bug during requirements and design phase". Barry Boehm, Victor R. Basili, IEE Computer, 34(1): 135-137, 2001

SECURITY ASSESSMENT OF COMPLEX SOFTWARE SYSTEMS (TENET #1)

Overview

The Business Threat Modeling process identifies, classifies and evaluates software vulnerabilities in order to recommend cost-effective countermeasures. The process is iterative and its steps can run independently, enabling any step to feed changes into previous steps even after partial results have been attained.



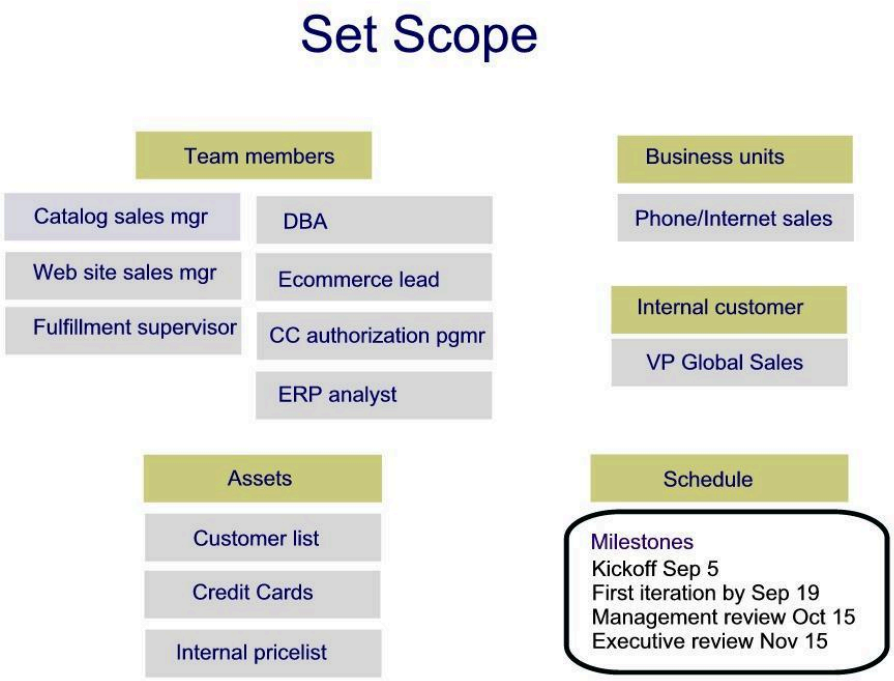
Continuous review of findings is key to success of the project. For example, an end-user may point-out fatal flows in an order entry form to the VP engineering during the `Validate Findings` step and influence the results in the `Classify Vulnerabilities` and `Build the threat model` steps.

1. Set scope.

The first step is to determine scope of work in terms of business units and assets. Focus on a particular business unit and application functions will improve the ability to converge quickly. The process will also benefit from executive level sponsorship that will need to buy into implementation of the risk mitigation plan.

The team members are chosen at a preliminary planning meeting with the lead analyst and the project's sponsor. There will be 4-8 active participants with relevant knowledge of the business and the software. The team is guided by expert risk analysts that have good people skills and patience to work in a chaotic process.

The output of `Set Scope` is one page:

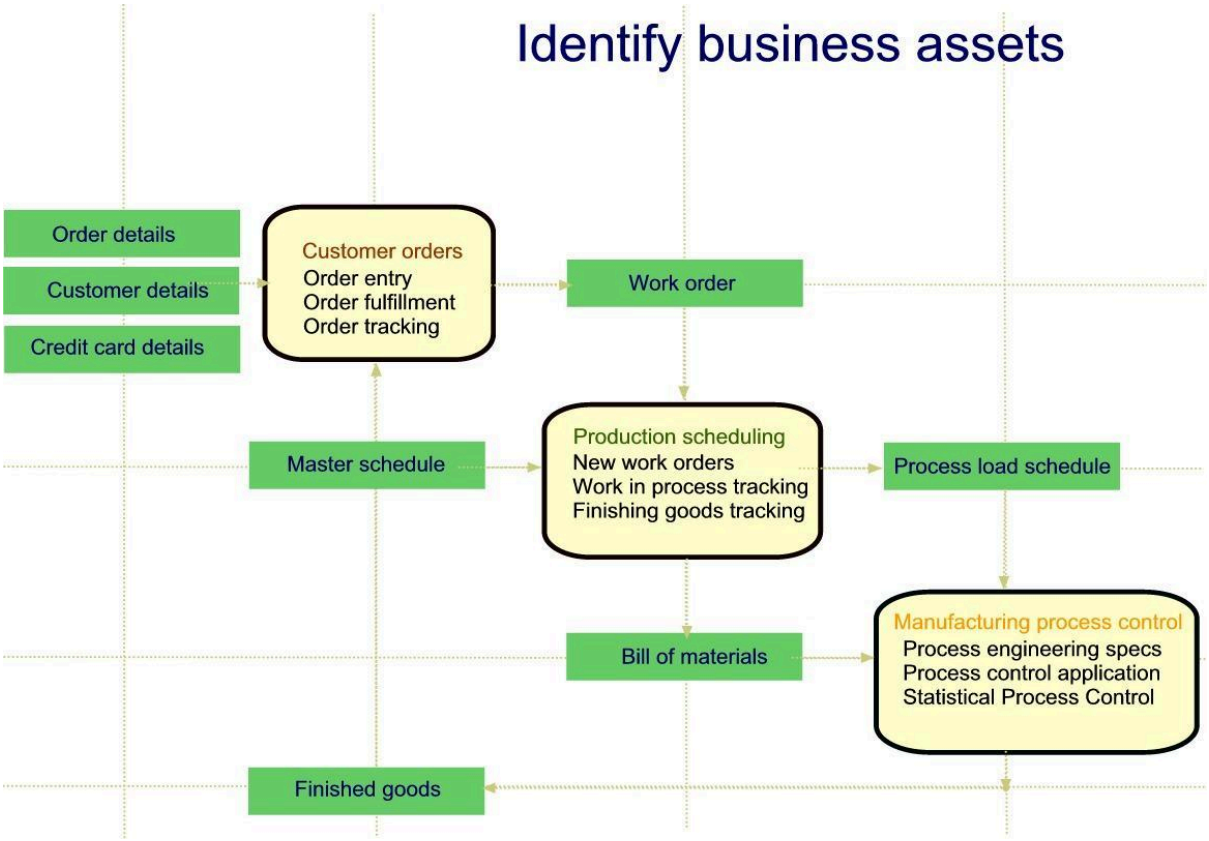


2. Identify business assets

In step 2, the team identifies operational business functions and their key assets:

This part of the process can be done using wall-charts as shown in the below figure. The graphic format helps the team visualize the scope of assets and estimate potential impact of threats on assets.

Business functions (white boxes) are placed on a diagonal from top left to bottom right as shown in the below figure. Assets (colored in green) flow clockwise around the diagonal of business functions.



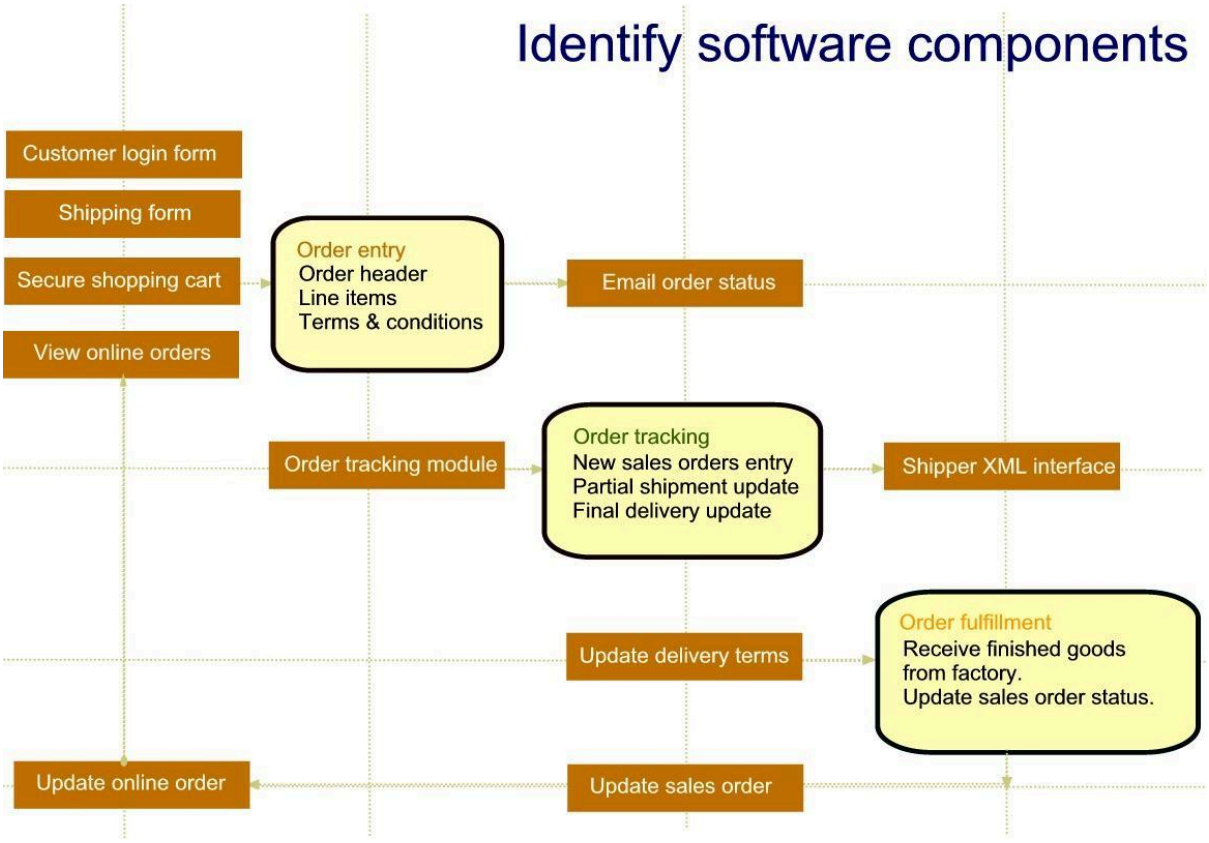
3. Identify software components

After identifying business functions in `Identify business assets`, the team now identifies software components (but doesn't assess vulnerabilities) using two sub-steps:

a. Identify application functions that serve the business function
b. Decompose application functions to software components

In order to help build a consistent, reasonably high-level view of the system, this part of the process can be done using wall-charts as shown in the below figure.

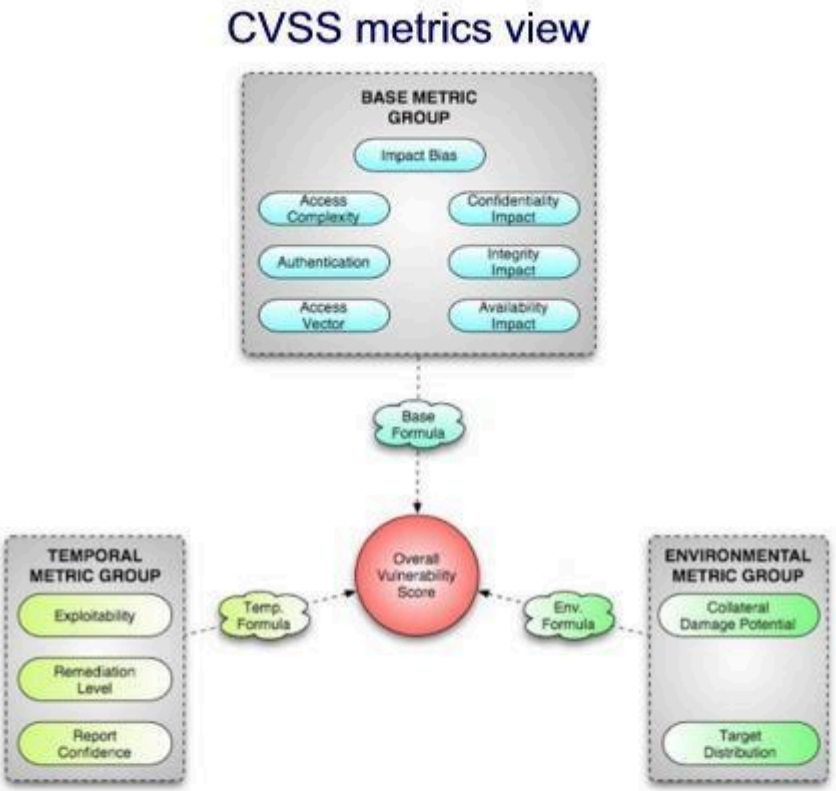
Application functions (white boxes) are placed on a diagonal from top left to bottom right as shown in the below figure. Decomposed components (colored in tan) flow clockwise around the diagonal of application functions.



4. Classify the software vulnerabilities.

CVSS[6] scores are computed for each component identified in the `Identify software components` step. In addition to the CVSS score, we collect an additional field, the CLASP [7] `problem type category`, for example **`"Use of hard-coded password"`**.

The knowledge base supporting the process contains a baseline of classified software vulnerabilities and evolves over time as the team classifies new vulnerabilities. Various source code scanners may also be used in this step, for example – `FindBugs` to find problems in Java source code.



Problem type category (example)

Use of hard-coded password
Severity: High
Likelihood of exploit: Very high
Avoidance and mitigation

```
Utilize a "first login" mode,
which requires the user to
enter a unique strong password.
```

[6] CVSS (Common Vulnerability Scoring System) is a standard way to convey vulnerability severity and help determine urgency and priority of response, http://www.first.org/cvss/intro/ Vendors such as Cisco, Symantec and Skype use CVSS to score their own application vulnerabilities.

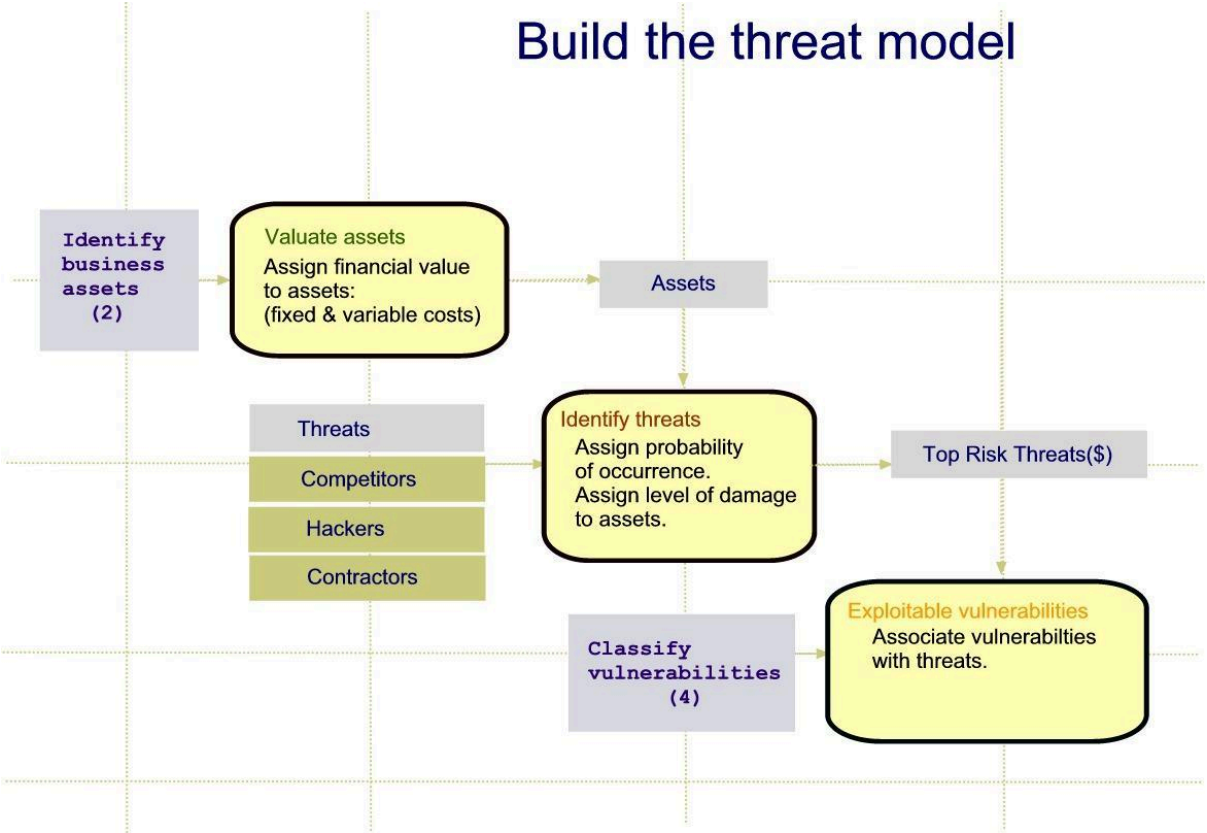
[7] CLASP (Comprehensive, Lightweight Application Security Process), http://www.owasp.org/index.php/CLASP

5. Build the threat model

The team now populates a quantitative threat model.

The threat model provides analytical capability to calculate risk and recommend the most cost-effective countermeasures based upon asset value, threat probability of occurrence, exploited vulnerabilities and percent damage to the assets during an attack.

Assets collected in the `Identify business assets` step are assigned a financial value. Threats are named and classified as to their probability of occurrence and damage levels. Vulnerabilities that were collected in the `Classify the vulnerabilities` step are associated with threats

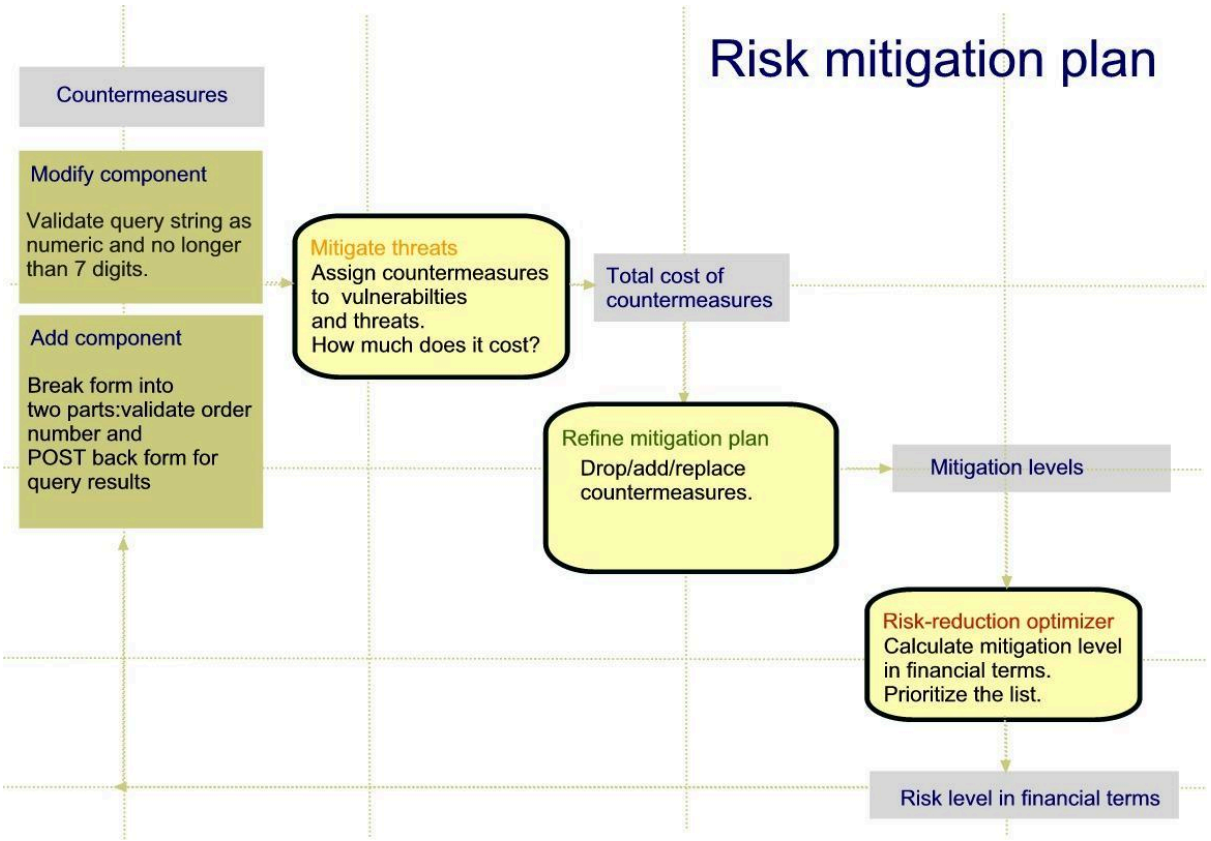


6. Build the risk-mitigation plan

In step 6, the team specifies countermeasures for vulnerabilities found in the software components and records them in the threat data model. While the best countermeasure for a problem is fixing it, in reality there may not be documentation and the programmers who wrote the code are probably in some other job. This means that other means may be required, such as code wrappers or application proxies.

The possible types of countermeasures are `Retain`, `Modify` and `Add` as seen in the below figure:

- `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6325992/Retain` the existing component (leave the defects in place) or,
- `Modify` the component (fix the defect or put in a workaround) or,
- `Add` components (for example call the Global LDAP directory to authenticate on-line users instead of using a proprietary customer table).
- Each `countermeasure` is assigned a cost and mitigation level. The cost may be a combination of fixed and variable cost in order to describe a one time cost of fixing a problem and ongoing maintenance cost.



7. Validate findings

This extremely important step validates the current findings with expert/relevant players in the enterprise.

The objective is to use all means at the disposal of the team to qualify components and vulnerabilities as to **where** (they are in the system), **which** (assets are involved), **what** (they do now and in the past), **why** (they perform the way they do) and **when** (a component is initialized and activated).

Conceptually, no limits are placed on what questions can be asked. Users may downgrade low-risk software components and escalate others for priority attention. They may add or remove assets from the model and argue parameters such as probability, asset value, estimated damage etc.

For example, a server-side order confirmation script that sends email to the customer may have received a low CVSS score in `Classify the software vulnerabilities`. The team can simply decide to eliminate that vulnerability from the list during `Validate findings`.

QUANTITATIVE EVALUATION AND FINANCIAL JUSTIFICATION (TENET #2)

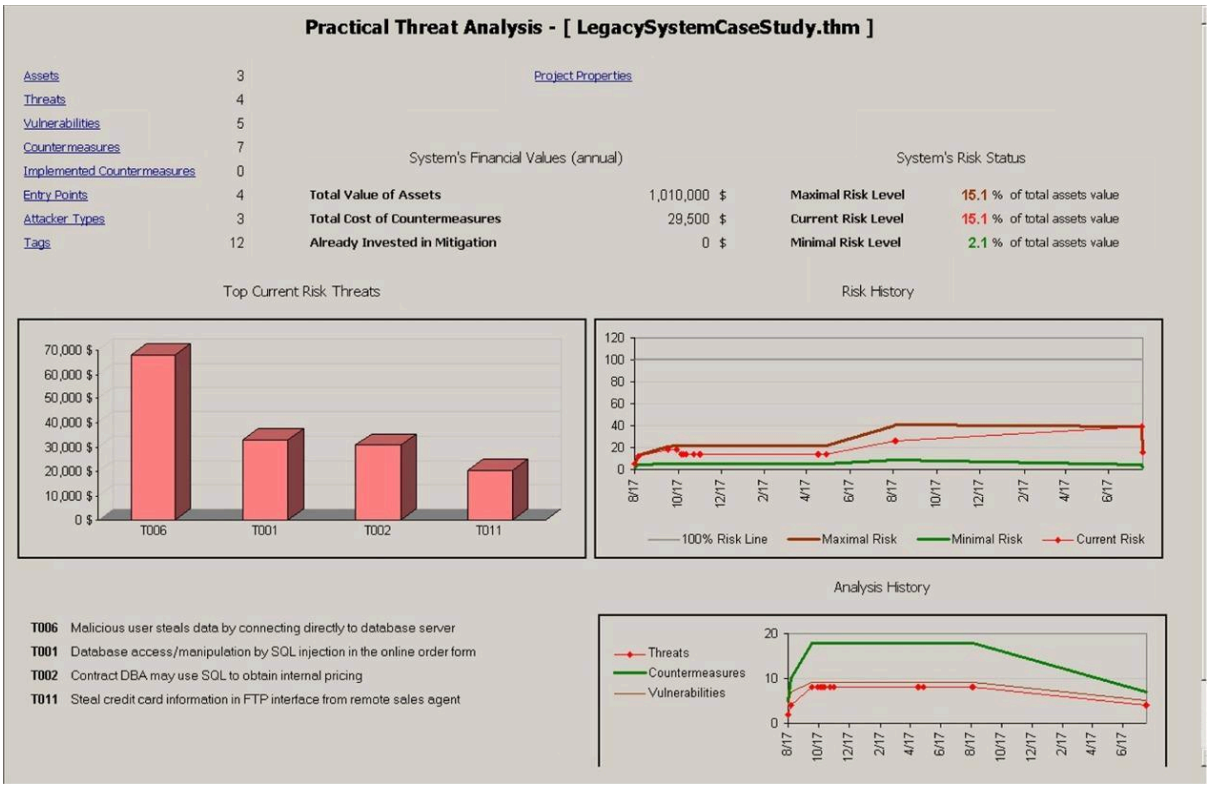
The output of the process provides executives with financial justification for an effective risk mitigation plan.

After specifying security countermeasures, the threat model risk-reduction optimization algorithm produces a prioritized list of the countermeasures in financial terms. The risk-reduction algorithm combines the most cost-effective countermeasures to reduce the overall system risk level to a minimum at a total fixed and variable cost.

Customer case study

We analyzed a business unit with systems for online order processing, make-to-order production and customer credit payment processing. Three key assets were identified: customer records, credit card details and internal price lists. As can be seen in the below financial analysis, the risk to the assets can be reduced from 15% to 2% at a cost of $29,500 with seven selected countermeasures. Risk is reduced to a minimum by proactive defect-reduction at a cost of less than 3 percent of the asset value. The cost is an order of magnitude less than acquisition of a proprietary system for preventing leakage of credit cards and privacy information.

You can see the Risk profile summary below. Click here to book a call with OpenCRO and see how the threat model is built from your threat scenario story.



EXPLICIT COMMUNICATIONS BETWEEN DEVELOPERS AND SECURITY (TENET #3)

The first order of business is having people talk to each other and argue the issues. By publishing CVSS scores and countermeasure costs, the developer and security teams can be confident that they can respond to a particular type of event in a consistent fashion.

We have found in our practice with clients that online collaboration tools (such as a Wiki) help give the company a clear, updated picture of well-known defects and security events. There is an abundance of such tools – and a discussion is outside the scope of this article.

What are the requirements?

An online collaboration tool should support *continuous* enterprise risk analysis and management. The tool should enable building a knowledge base and tracking open issues.

1. The knowledge base contains standard CVSS scores for components and CLASP problem types classifications are always available for the entire organization. Users can add new entities and modify scores as the business environment changes.
2. The issue tracker provides:
    - A consistent thread of requests, changes and open action items during the risk analysis process and in particular in the `Validate findings` step
    - Updated implementation status of countermeasures.
    - Unlike email, issues cannot get lost or be ignored!

Afterword

SUSTAINING CONTINUOUS RISK REDUCTION

Training a team that can sustain quality

While the Business Threat Modeling process itself has educational benefits, there is no question that the quantity and complexity of production systems in a large organization requires skills for continuous risk analysis and defect reduction.

To meet this objective, we'd suggest 2 training programs for software development teams.

The first - "*Business Threat Modeling for the enterprise*" is a two day course that trains analysts and qualifies them to run a defect reduction process as described in this paper. An advanced course would qualify analysts, who have conducted several risk analysis projects, to teach the "*Business Threat Modeling for the enterprise*" course and coach others.

Improving best practices in the software development life cycle

The risk analysts supporting the process, together with the knowledge base are a significant resource for any organization that wants to evaluate the economic feasibility of a defect reduction program and improve best practices in the software development life cycle:

1. How to reduce avoidable rework
2. How to reliably identify fault-prone modules in a company's particular operation
3. How to identify modules with the most impact on system reliability and downtime
4. Develop sustaining metrics for defect reduction
5. Train application programmers in best security practices and help them see themselves as part of an integrated company-wide commitment to quality software.
6. Help the organization choose and implement disciplined practices such as Watts Humphrey's PSP (Personal Software Process) and TSP (Team Software Process) that can have high ROI in defect reduction in new software development.