

Making secure software

What threats really count?

ABSTRACT

What threats really count for your business?

No question is more important for implementing an effective security and compliance program for your business. The management, the software developers and security analysts cannot expect to mitigate risk effectively without knowing the sources and cost of threats to the organization.

A modern organization depends on its software applications in order to conduct business. The prevailing security model predicated defense in depth of these systems. The most common strategies are to mitigate external threats 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 your business? The answer is clearly no.

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

This paper presents a rigorous software development process for delivering secure software using a simple notion – “buggy software is insecure software”. By removing software defects we are in the best position to deliver secure software to our customers.

WHY DEFECTIVE SOFTWARE IS INSECURE SOFTWARE

There are 3 reasons why defective software is at the core of security breaches.

1. The empirical data says that software quality matters

This evidence graphically emerged from a study that analyzed a sample of 167 customer data breaches 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.

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%
Software defects maliciously exploited.	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

2. Most companies are not committed to improve their 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 because:

- They usually receive conflicting proposals for new security initiatives with weak or missing financial justifications.
- 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://infosecurymag.techtarget.com/articles/1999/paller.shtml>

3. Security defenses don't improve our understanding of the root causes of data breaches

Why is this so?

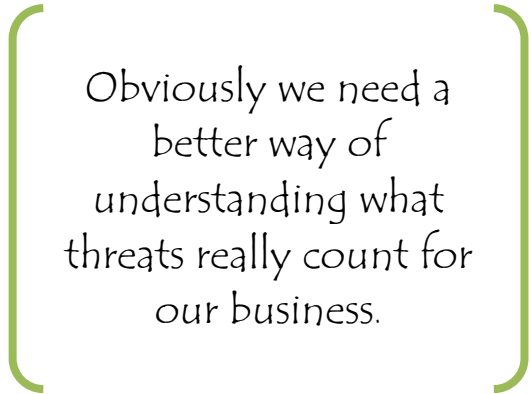
First of all – defenses are by definition, not a means of improving our understanding of strategic threats. Think about the Maginot Line in WWI or the Bar-Lev line in 1973. Network and application security products that are used to defend the organization are rather poor at helping us understand and reduce the operational risk of insecure software.

Second of all - it's hard to keep up. Security defense products have much longer product development life cycles than the people who develop day zero exploits. The battle is also extremely asymmetric – as it costs millions to develop a good application firewall that can mitigate an attack that was developed at the cost of three man months and a few Ubuntu workstations. Security signatures (even if updated frequently) used by products such as firewalls, IPS and black-box application security are no match for fast moving, application-specific source code vulnerabilities exploited by attackers and contractors.

Remember – that's your source code, not Microsoft.

Third – threats are evolving rapidly. Current defense in depth strategy is to deploy multiple tools at the network perimeter such as firewalls, intrusion prevention and malicious content filtering. Although content inspection technologies such as DPI and DLP are now available, current focus is primarily on **the network**, despite the fact that the majority of attacks are on **the data** - customer data and intellectual property.

The **location** of the **data** has become less specific as the notion of trusted systems inside a hard perimeter has practically disappeared with the proliferation of cloud services, Web 2.0 services, SSL VPN and convergence of almost all application transport to HTTP.



Obviously we need a better way of understanding what threats really count for our business.

SO IF IT'S SO IMPORTANT – HOW DO YOU ACHIEVE QUALITY SOFTWARE?

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

1. Users of modern software development methodologies (including Extreme Programming and Agile) may overlook threat analysis of their software in favor of rapid deliveries to customers in order to get feedback that will improve the product and its business model. Unfortunately – there are business domains such as medical devices and healthcare, where a bug can literally kill you and your company.
2. The cost of finding and fixing bugs in production systems is regarded as too high.⁵
3. Software developers and security teams generally don't talk to each other. The bigger the company, the harder they fall, as information gets lost in the cracks.

We propose establishing three principles

1. **Know your software's mission.**
Collect data from everyone involved in the software development, delivery and support process and classify defects for risk mitigation according to standard vulnerability and problem types.
2. **Prioritize**
Quantify the risk in terms of assets, software vulnerabilities, and the organization's current threats.
3. **Talk**
Explicit communications between software and security teams is essential to translating security analyst knowledge of vulnerabilities into defect reduction of the software and threat mitigation.

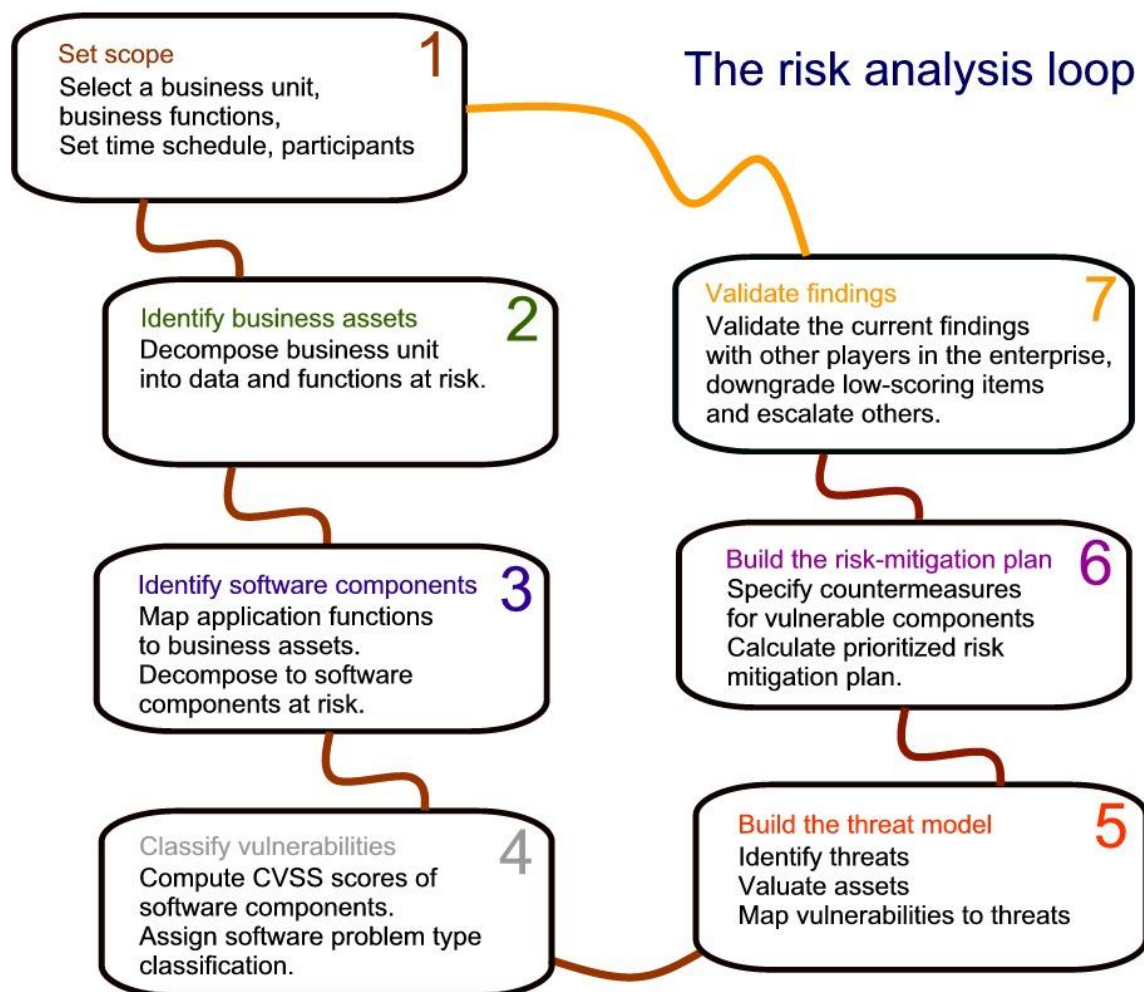
⁵ "In production, it's often 100 times more expensive than finding and fixing the bug during requirements and design phase". Barry Boehm, Victor R. Basili, IEE Computer, 34(1): 135-137, 2001

KNOW YOUR SOFTWARE'S MISSION

Overview

Knowing the mission of your software is the first step. Is it a medical device that must comply with FDA and HIPAA regulations? Is it a Web 2.0 service for airline reservations that must be fast and friendly? Is it a financial analysis application that must be precise?

The 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 the 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.

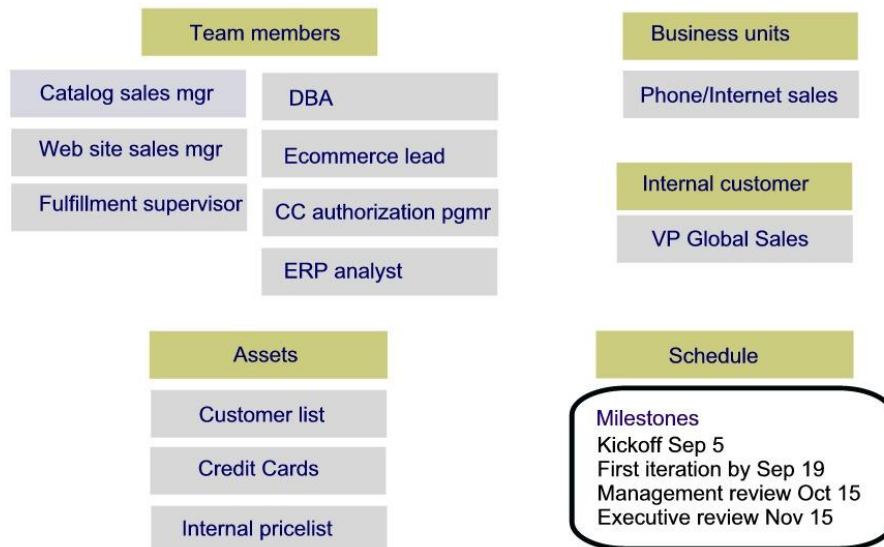
1. Set scope.

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:

Set Scope

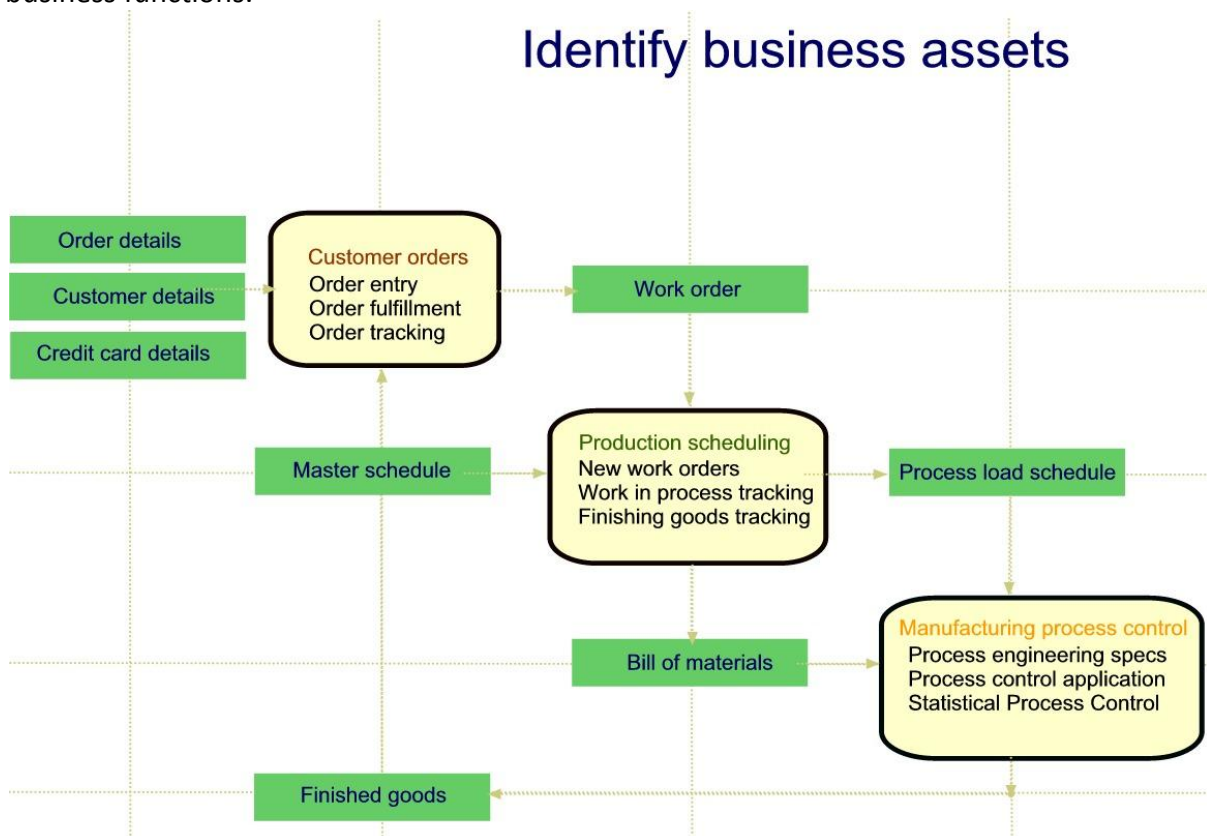


2. Identify business assets

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.



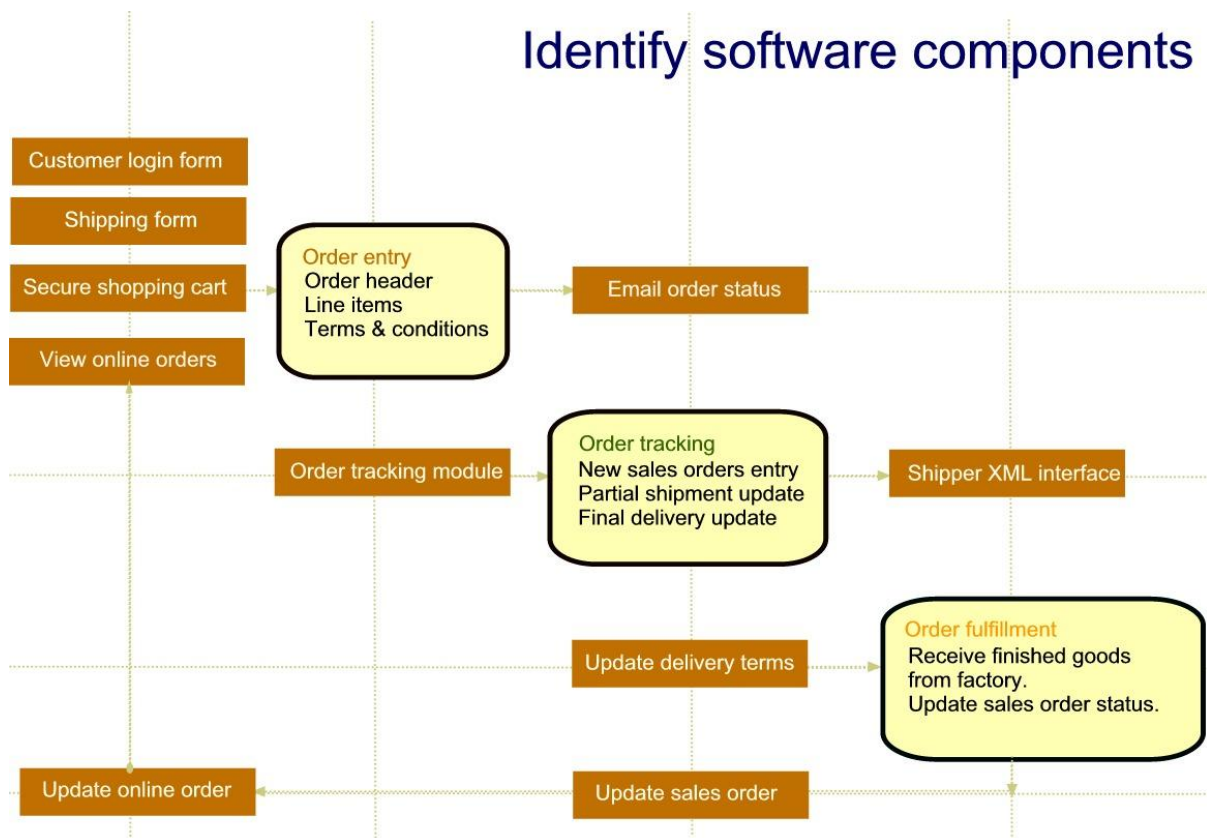
3. Identify software components

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

- Identify application functions that serve the business function
- Decompose application functions to software components

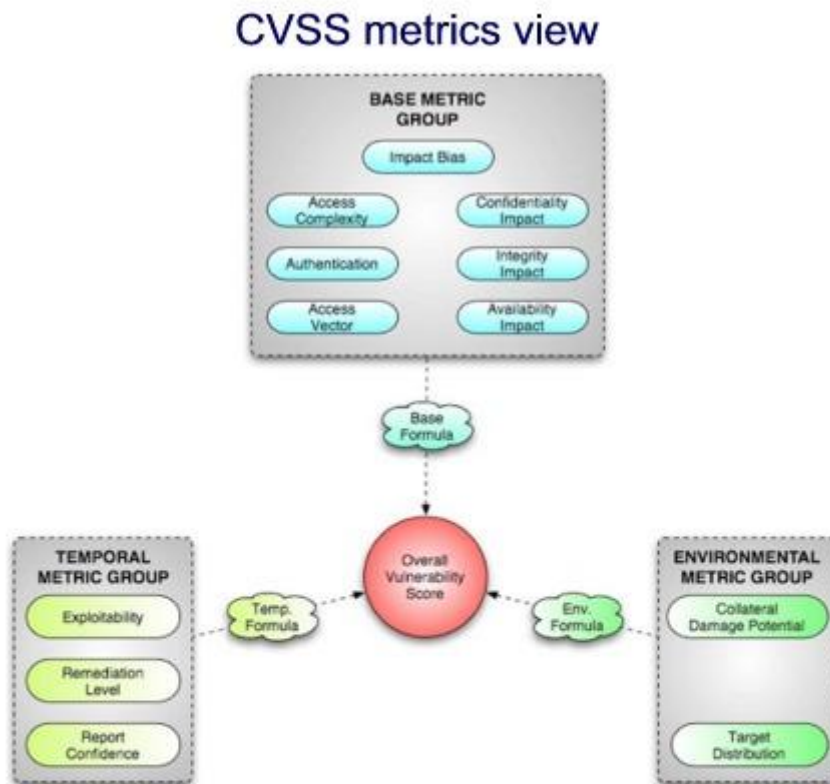
In order to help build a consistent, reasonably high-level view of the system, this part of the process can be done using wall-charts as shown in the below figure.

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.

Use metrics. We suggest computing CVSS⁶ scores for each component identified in the Identify software components step. In addition to the metrics, we collect an additional field, the CLASP⁷ problem type category, for example: "Use of hard-coded passwords". Using metrics creates a baseline of classified software vulnerabilities which evolves over time as the team classifies new vulnerabilities. Various source code scanners may also be used in this step.



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.

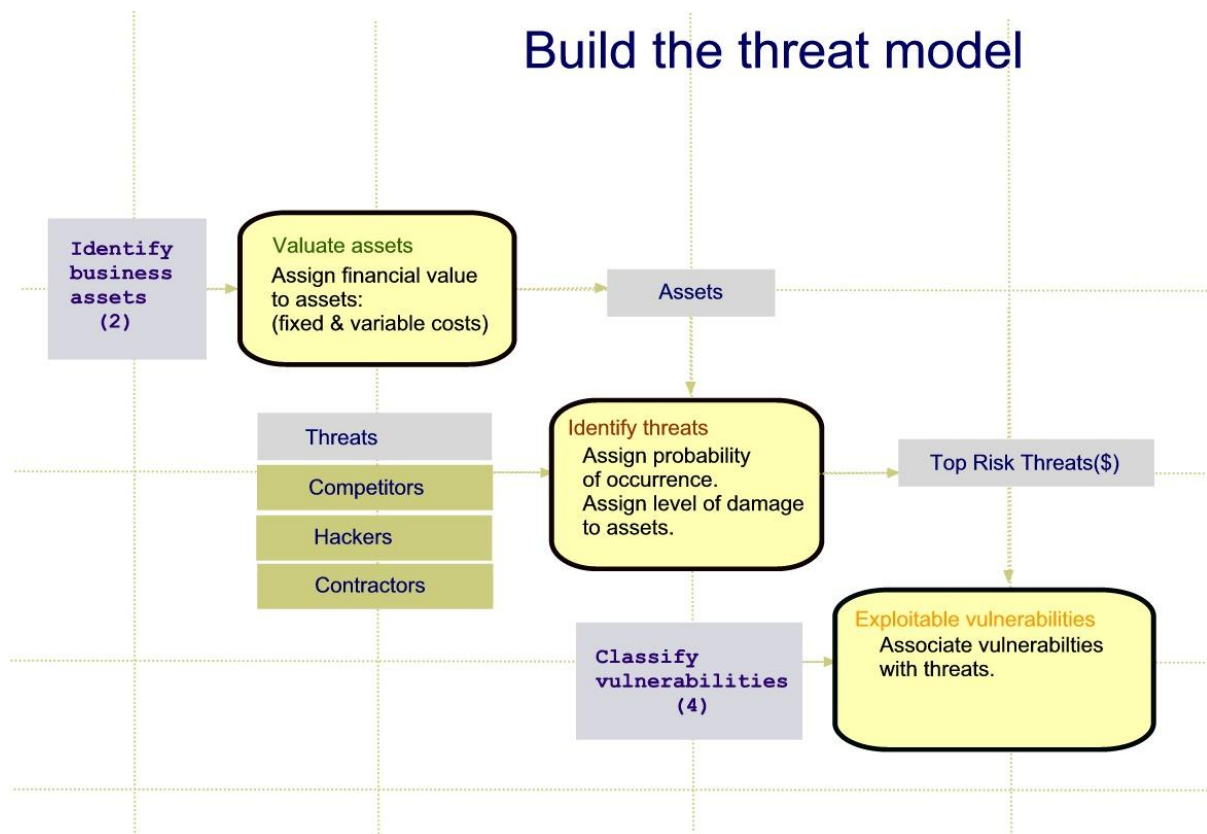
⁶ CVSS (Common Vulnerability Scoring System) is a standard way to convey vulnerability severity and help determine urgency and priority of response, <http://www.first.org/cvss/intro/>. Vendors such as Cisco, Symantec and Skype use CVSS to score their own application vulnerabilities.

⁷ CLASP (Comprehensive, Lightweight Application Security Process), <http://www.owasp.org/index.php/CLASP>
Copyright 2006 Danny Lieberman This work is licensed under the Creative Commons Attribution License.

5. Build a threat model

It's now time to build a threat model of the system.

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

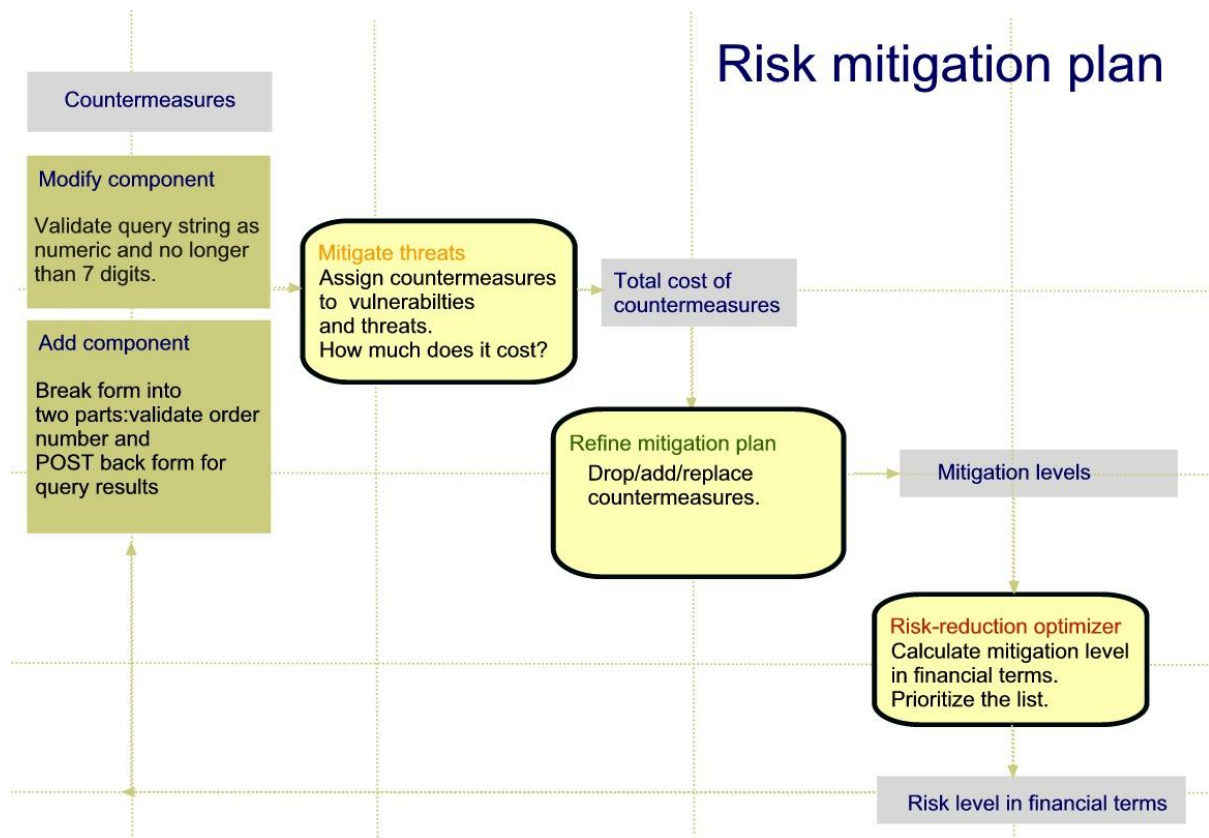


6. Build a threat-mitigation plan

In step 6, the team specifies countermeasures for vulnerabilities found in the software components and records them in the threat 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:

- 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 use an LDAP directory service 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 onetime cost of fixing a problem and ongoing maintenance cost.



PRIORITIZE

This critical step takes the current findings and involves the users, software developers and security analysts in prioritizing their objectives for defect reduction.

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. **End 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 during the Classify the software vulnerabilities step. The team can simply decide to accept order confirmation script as it is and focus on larger targets such as possible back doors into the database.

TALK

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 for version management such as Subversion and GIT are essential for the company to have a clear, updated picture of well-known defects and security events.

There is no question that the complexity of implementing secure software applications requires continuous threat analysis and defect reduction.

To meet this objective, we propose training and qualifying a team member to become the threat analyst who will oversee the defect reduction process as described in this paper.

The analyst supporting the process, together with a knowledge base of defects are a significant resource for any organization that wants to improve their software development and deliver secure software.