Three Ways to Simplify Auditing Software Security Requirements and Design

It is common knowledge that building security into software is an important prerequisite for information assurance. Besides being 30 times cheaper to fix a defect in design versus fixing it after the fact, several IT control frameworks and regulations suggest or mandate the use of security requirements and design. Most auditors have a tough time assessing these controls due to a lack of artifact-based, available evidence, nor can they offer guidance to development teams on how to produce such evidence. Auditors generally rely on interview techniques and the existence of policies to make assessments. This approach leads to development teams downplaying the importance of, and often totally ignoring, security requirements and design, even though these controls are critical enough to warrant being part of several compliance frameworks.

Taking one example in more detail, the Payment Card Industry Data Security Standard (PCI DSS) section 6.3 states, “Develop internal and external software applications (including web-based administration access to applications) securely… Incorporating information security throughout the software development life cycle.” The testing procedures include examining written processes and interviewing development team members to ensure that the procedures are, in fact, being followed. Section 6.5 states, “Prevent common coding vulnerabilities in software-development processes as follows… Develop applications based on secure coding guidelines.” The testing procedures again reference examining policies and procedures, interviews, and an additional reference to examining training records. It is imperative for auditors to ask for better evidence—documents or other artifacts—that prove security was incorporated into system requirements and design for each application. High-level requirements, such as “make the system secure” or “provide sufficient authentication and access control,” are not sufficient. Much like using the Open Web Application Security Project (OWASP) Top 10, vague general requirements do very little to ensure that sufficient controls are built into application design.

Fortunately, advances in technology and tool support give auditors a few different options to easily move beyond simple policy and interview-based assessments. These approaches are not necessarily mutually exclusive. Many organizations use a combination of approaches. However, auditors should seek evidence from at least one of these techniques.

THREAT MODELING APPROACH

Threat modeling is a technique of modeling an application’s design to uncover potential threats based on a systematic, repeatable process. Developers and security teams prioritize the resultant list of threats, along with corresponding countermeasures, and use these to incorporate security into the software design. Microsoft has been the biggest champion of threat modeling, along with extensive freely available documentation.

Microsoft began championing threat modeling as part of the broader Trustworthy Computing initiative, after which other organizations have proposed their own version of threat modeling, e.g., the OWASP Application Threat Modeling methodology (figure 1).

Not surprisingly, Microsoft also has a free tool that it has released to help implement threat modeling. MyAppSecurity also offers a threat modeling tool. Both tools allow development teams to provide evidence that they have incorporated security into software design.

Threat modeling is the most comprehensive of the three approaches. Done correctly, threat modeling can reveal an exhaustive list of all potential security issues within an application and drive holistic defensive approaches. Its incorporation of data flow diagrams also allows
development teams to understand not only what their security concerns are but also where defensive controls should fit with respect to system components. Conversely, threat modeling’s comprehensiveness is also a shortcoming for many organizations. Although most tools can be used without information security expertise, in most cases, proper threat modeling requires people with security experience to determine comprehensive threats. This is challenging due to a global shortage of information security expertise. It also requires certain documentation, such as architecture diagrams, while increasingly agile teams adopt the mentality of working software over comprehensive documentation.

From an auditor’s perspective, a documented threat model shows clear evidence of application security being incorporated into software design. For organizations that adopt threat modeling, auditors should seek to review standard output from threat models for specific applications. Examples include:

- A documented list of threats along with appropriate countermeasures
- A data-flow diagram illustrating processes and trust boundaries
- A comprehensive audit should include examination of the details of these documents, but some auditors may not have a sufficient technical background to perform a thorough review. In such cases, auditors may wish to examine the following through interviews and basic artifact examination:
  - Was the threat model documented within the time period in question (e.g., current financial year)?
  - Was the threat model uniquely generated for the particular application in question? A generic threat model applied to multiple applications is of little value, as the threats to each application are unique.
  - Were the identified countermeasures adopted into application design, and, if so, is there any evidence to support this (e.g., tickets in bug tracking tools, email trail, meeting minutes)?
  - Were any threats deemed to be accepted risk or were they all mitigated? If threats were accepted, who accepted the risk and is there any audit trail to support this?
THE CONTROLS LIBRARY APPROACH

The emerging ISO 270341 application security standard from the International Organization for Standardization (ISO) outlines a process of defining application security controls systematically across the organization. In layman’s terms, it requires organizations to define a library of common software security controls (figure 2). It then requires each application team to select a subset of these controls based on a variety of business, regulatory and technological factors. Developers assert their conformance to the applicable controls during development, and another party (often security or quality assurance [QA]) verifies that the controls are in place. While some organizations may not have plans to comply with ISO 27034, the standard serves as a useful reference to anyone planning to create an application security program. ISO 27034 has evolved with participation from many industry stakeholders. Several organizations with strong application security maturity have naturally built a similar approach over time.

The controls library approach is easily implemented within the context of a software development process, with the entire process generally taking between two to four hours when leveraging automation. Implemented correctly, it can also generate a robust set of requirements to address most well-

---

**Figure 2—Example View of a Controls Audit Tool**

<table>
<thead>
<tr>
<th>10</th>
<th>T136: Do not store sensitive credit card data.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sensitive credit card data must not be stored.</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="2 Notes" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10</th>
<th>T68: Encrypt credit card PANs in storage.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Render primary account number (PAN) unreadable anywhere it is stored (including on portable) by using any of the following approaches:</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="2 Notes" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10</th>
<th>T21: Ensure confidential data are sent over an encrypted channel.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Confidential data must always be sent over an encrypted channel for a security-sensitive application version 1.2 to establish a secure channel. Enforce this requirement by explicitly refusing plain-portions for an application.</td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="1 Notes" /></td>
</tr>
</tbody>
</table>

known, preventable software security issues. It is, thus, a middle ground between being comprehensive and lightweight. It is not as comprehensive or potentially accurate as threat modeling. Moreover, it does not help inform developers where security controls should fit within an application's architecture. It generally requires more time to implement upfront and maintain than something as lightweight as a security checklist.

Organizations that adopt a tool for following the controls library approach should be able to generate an audit report that documents all appropriate requirements and their status in terms of development and verification. Auditors should make note to examine the following:

- Is the set of business, technological and regulatory factors used to decide upon the security controls documented?
- Was each control implemented and verified? If any control was not implemented or verified, is there audit evidence that it will be implemented/verified later or are the risk factors accepted?
- If applicable, is there evidence that the controls were integrated into development, such as linkage to an application life cycle management (ALM) tool (e.g., JIRA)? Organizations may elect to document completion status within the controls library tool itself or they may elect to use an ALM tool instead.

THE SECURITY CHECKLIST APPROACH

Another common approach to improving application security is to provide comprehensive checklists that enumerate all known threats and corresponding countermeasures. At its most basic form, organizations often build a static, secure programming guide to accomplish this. There are, however, several challenges with a large static checklist or guide:

- **Time pressure**—Developers under time pressure to deliver a feature, iteration or release rarely have time to sift through a 40-plus page document looking for best practice guidance.
- **Seniority**—Senior developers can often feel they already have sufficient expertise in security, often ascribing security problems to more junior developers. It is natural for them to feel skeptical that any general best practice guidance can really benefit their application.
- **Static content**—A single document can quickly grow out of date with advances to attacks and defensive technologies. Developers need actionable information relevant to today’s threats.

- **Context switch**—Studies show that developers lose productivity every time they shift context out of their development tools. Asking developers to switch between their regular tools and a static document means lost productivity, which, in turn, reduces the likelihood of the document being read.

Fortunately, automated tools that integrate with development environments help reduce the burden of having to parse large documents. Security Innovation’s TeamMentor product provides a dynamic, tool-based method for secure programming. It offers much more functionality than a standard checklist but is just as easy to use and implement.

Security checklists are the lightest weight of all three methods. On the other hand, they are often not uniquely tailored to an application like the threat modeling and application security controls approaches.

Auditors should ask to review the following:

- A copy of the completed checklist that the development team used
- An audit trail of who completed the checklist
- Audit evidence that the checklist items were integrated into development
- An understanding of which specific checklist items were not implemented and if they were simply not applicable or if they were accepted risk

<table>
<thead>
<tr>
<th>Figure 3—Table Summarizing Three Approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Approach</strong></td>
</tr>
<tr>
<td>Threat modeling</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Controls library</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Checklist</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Overall, organizations have several tools and techniques at their disposal to incorporate security into requirements and design. The three approaches are not necessarily alternatives (figure 3). Several organizations use two or all three of the approaches described. Relying on process documentation and interviews alone to assess these controls is no substitute for real evidence that the process has been followed during application development. Since many organizations form their information security programs primarily to address audit requirements, the consequence of lax audit requirements means many software development teams place very little emphasis on software security requirements and design. Advances in automation allow auditors to build trust, but verification via reviewing the real artifacts that prove development teams are building in security is still needed.

ENDNOTES


2 A nonexhaustive list includes: ISO 27001:2013 sections A.14.1.1 and A.14.2.5; PCI DSS sections 6.3 and 6.5; FFIEC IT Handbooks, Security Controls Implementation Systems Development, Acquisition, and Maintenance Software Development and Acquisition; COBIT® 4.1: A12 Acquire and Maintain Application Software; COBIT® 5: BAI02 and BAI03.09; NIST 800-37: Common Control Identification Task 2-1 and Security Control Selection Task 2-2; and NIST 800-55: SA-15, SA-17


