SMARTBEAR
Collaborator

Whitepaper

How to Minimize Risk:

The Payment Card Industry and Staying Ahead of Regulatory Compliance

SMARTBEAR

Retailers are among the top targeted vertical markets for cyberattacks.

The retail industry is particularly attractive to hackers and security breaches, largely because most retailers process customer credit and debit card information through their systems.[1]

In recent years, a data breach at Home Depot affected 56 million credit card accounts and 53 million email addresses, and the cost to the company was an estimated at $80M before insurance reimbursements. Hackers gained access to the company's computer network using stolen account information from a third-party vendor.[2]

Stories like that are a nightmare for companies. The fiscal damage is bad enough, but who knows how much damage is done to the brand, and for how long. It can make people lie awake at night asking, "Did I do everything possible to ensure our software is as good as possible?"

Viruses, malicious code, password cracking, social engineering, and other threats that are common to most business networks must be considered in designing PCI-related software systems. The Common Weakness Enumeration (CWE) community and the SANS Institute collaborated to identify and list the Top 25 Most Dangerous Software Errors.

The study examined critical systems across several highly targeted industries. The list ranks the most widespread and critical errors that can lead to serious vulnerabilities in software. They are often easy to find, and easy to exploit.

These top 25 are dangerous because they frequently allow attackers to completely take over the software, prevent the software from working, or just plain steal data.[1] If these errors are easy to find and exploit, why are they not caught before the software is released?

The problem is complexity

As systems become more capable, it becomes harder to test all the ways they will be used. Once you test software and fix all the problems found, the software will always work under the conditions for which it was tested.

Test-and-fix approaches are vital, dynamic testing methods. Whether performed on individual units or the entire system, dynamic approaches that test the code in action share one common shortcoming: they all rely on test cases.

Test case scenarios are constructed from the same source documents that developers use, such as requirements and specification documents. These documents are much more comprehensive at defining what the finished product should do, rather than what it shouldn't do. Developers inject about 100 defects into every 1,000 lines of the code they write.[2]

Seemingly insignificant changes in software code can create unexpected and very significant vulnerabilities elsewhere in the software program. Many of these defects will have no impact on the test case scenarios designed for testing. Yet, they could have devastating, unforeseen effects in the future.

If quality can't be tested, then what?

Software quality assurance should focus on preventing the introduction of defects into the software development process, rather than trying to 'test quality into' the software code after it is written. Where testing methods fail, the best approach is direct examination of the code and related artifacts.

Examination activities typically include:

- **Walk-throughs and peer reviews**
- **Automated code analyses**
- **Code and document inspections**
- **Module-level testing**
- **Integration testing**

Code walk-throughs and peer reviews are systematic examinations of the source code. Code reviews can often find and remove common vulnerabilities that dynamic testing miss – such as format string exploits, race conditions, memory leaks and buffer overflows, which lead to security and functionality problems.

While some believe analysis of the code is best done by automated tools, code reviews are actually more effective at finding errors than automated tools. Most forms of testing only average about 30% to 35% in its defect removal efficiency levels, and seldom top 50%. Formal design and code inspections, on the other hand, can achieve 95% in defect removal efficiency.[3]

Peer reviews give you proof of compliance for audits

Measuring defect removal is critical to proving compliance. The challenge, whether done through dynamic testing or direct examination, is that a developer cannot test forever, and it is hard to know how much evidence is enough. Measures such as the number of defects found in specifications documents, estimates of defects remaining, testing coverage, and other metrics are all used to develop an acceptable level of confidence before shipping the product.

While safety is the primary measure, the process should also be sufficient to prove compliance in an audit or litigation scenario. "The important point is that during the software design and development process, think in terms of risks associated with the application, thus reasonably anticipating threats to the security of the application."[4]

Code reviews should be in writing, online, and indexed for easy retrieval – otherwise, there is no proof they have been performed. Statements about the code in general, specific lines and specific issues should all be tied to the person, time, and date of their identification. If needed, this data should be presented as both comments and metrics to allow for accounting of the development process.

Source code evaluations should be extended to verification of internal linkages between modules and layers (horizontal and vertical interfaces) and the compliance with their design specifications. Documentation of the procedures used and the results of source code evaluations should be maintained as part of design verification.

Code reviews should be supported by document reviews. Teams should review user stories, test plans, and other artifacts as part of the review process, and flag regulatory issues in the review to ensure they are considered.

> *Code reviews should be in writing, online, and indexed for easy retrieval — otherwise, there is no proof they have been performed.*

While thousands of organizations have successfully implemented and defended peer code reviews successfully, many have failed. The difference often comes down to poor implementation strategies. And they are all issues that can be readily addressed:

- **Reviews are too long.** After just a few hours, attention wanders. All-day code reviews can seem almost painful. Keep reviews short and no more than one or two hours per day. Developers can review between 150 and 300 lines of code, depending on complexity. Not surprising, this rate of review also provides the highest rate of defects identified per line of code (defects / LOC).

- **Reviews are seen as an additional task.** Especially when a review backlog builds up. Rather than let them become a bottleneck, make reviews a daily activity or take them as they come in.

- **Comments are seen as subjective.** It is easy to discount a colleague's comments and disregard their opinion. Make it easy for reviewers to annotate the specific code in question and to get other reviewers to weigh in.

- **Remote reviews can be challenging.** Distributed teams are a given, especially post-covid, and bringing teams together for reviews goes against the need for regular, brief reviews. Instead, facilitate remote reviews with tools designed for remote collaboration.

- **Documentation is not automated.** The administrative burden of documenting, archiving, and distributing this living document can be overwhelming. Use tools that make compliance documentation an automatic by-product of the review.

Companies successful with adopting code review facilitate the needs of developers first, then let the needs of the project and company naturally follow.

One of the most important contributions a company can make toward the successful adoption of code reviews is the set of tools it. The right tools enable each development team to find its own best method for code reviews, enabling a bottom-up approach to code review design and potentially gaining in quality and regulatory compliance.

Look for these characteristics in a code review tool:

1. ***Support team-designed rules and processes.*** Teams should be able to determine review intervals, workflows, and specific tasks to accomplish during the review while the tool supports and manages adherence.

2. ***Support each team's preferred mode of interaction.*** Whether it is side-by-side, remote real-time, asynchronous, or a combination, the team should decide. The tool should support before and after views of code and document changes and threaded contextual chat with references to files and line numbers.

3. ***Support for multiple IDEs.*** To make reviews a "normal" part of developers' work routines, developers should not need to leave their "regular" development environment to review code. Nor should the team need to change code review tools if they change IDEs in the future.

4. ***Provide seamless integration with SCM systems.*** To start reviews easily and expedite them, developers should be able to point to the code that needs review and have those files extracted automatically. Tools add tangible value to this process by automating the collection and distribution of these files.

5. ***Ensure documents are integrated within the review process.*** A standardized peer review process enables all project-related documents (e.g. PDF, MS Office, HTML, images, schematics, intranet and web-based document management system) to be reviewed the same way, making document reviews less frustrating for developers.

6. ***Enable accurate reporting.*** Meaningful metrics play a critical role in the reporting process to indicate progress and current status. Useful metrics used in meeting review milestones and audit requirements include man-hours spent in review, defect data, and lines of code inspection, as well as review approval and electronic signature status.

Different Methodologies, Different Paper

It should be noted that this paper has steered away from discussing any particular software development methodology. A peer code-review process can be implemented within waterfall, Agile and other methodologies with equal success. The point here is that, not only will implementing peer code reviews make the products your company produces better, it will make the processes and the people that produce them better as well.

Code reviews are a powerful tool for eliminating defects, but achieving compliance can be burdensome. Even in organizations where code reviews have been "adopted," they are skipped as much as 30% of the time, primarily because of inadequate support.[5]

Too often, organizations believe they can have ad-hoc development processes, and then use an inspection procedure at the end to remove all defects. That will not happen. Industry statistics indicate that for every four errors pulled out, one new error is injected. Therefore, only portions of defects are actually removed at the end of the implementation process. To approach zero defects, inspection must be an iterative process.[6]

For years, it was believed that the value of inspections is in finding and fixing defects. However, in examining code inspection data, it becomes clear that inspections are beneficial for an additional reason. They make the code easier to understand and change.

> *A peer code-review process can be implemented within waterfall, Agile and other methodologies with equal success.*

An analysis of data from a recent code inspection experiment shows that **60% of all issues raised in code inspections are problems that would not have been uncovered by latter phases** of testing or field usage because they have little or nothing to do with the visible execution behavior of the software.

Rather, they improve the maintainability of the code by making the code conform to coding standards, therefore minimizing redundancies, improving language proficiency, improving safety and portability, and raising the quality of the documentation — benefits not possible through automated testing.[7]

In conclusion

The goal of regulatory compliance is to minimize risk for the consumer, but it can benefit the software organization as well. A compliance strategy that utilizes peer code reviews creates an environment of shared understanding and collaboration. As developers review and comment on each others' code, they all improve. In the end, code review provides a platform for continuous process improvement, leading to better standards, better developers, and enhanced efficiency.

So while a systematic code-review process can help the organization prove compliance, it will certainly lead to higher quality finished product – which can help avoid a security event in the first place.

Citations:

1. "NTT Security intelligence report finds organizational attacks becoming more sophisticated," SecurityInfoWatch, January 25, 2017.

2. "11 Data Breaches That Stung US Consumers," Bankrate.com

3. "2011 CWE/SANS Top 25 Most Dangerous Software Errors," September 2011

4. "The Software Quality Challenge," Watts S. Humphrey, CrossTalk, June 2008

5. "Measuring Defect Potentials and Defect Removal Efficiency," Capers Jones, CrossTalk, June 2008

6. "Application Security – Next Layer of Protection," October 13, 2003, Keith Pasley, CISSP, developer.com

7. Mario Bernhart, Andreas Mauczka, Thomas Grechenig Research Group for Industrial Software (INSO) Vienna University of Technology, Austria, 2010

About SmartBear

At SmartBear, we focus on your one priority that never changes: quality. We know delivering quality software over and over is complicated. So our tools are built to streamline your DevOps processes while seamlessly working with the products you use – and will use. Whether it's TestComplete, Swagger, ReadyAPI, Cucumber, Zephyr, Bugsnag, or one of our other tools, we span from test automation, API lifecycle, collaboration, performance testing, test management, app stability and error monitoring, and more. Whichever you need, they're easy to try, easy to buy, and easy to integrate. We're used by 16 million developers, testers, and operations engineers at 32,000+ organizations – including world-renowned innovators like Adobe, JetBlue, FedEx, and Microsoft. Wherever you're going, we'll help you get there. Learn more at smartbear.com, or follow us on LinkedIn, Twitter, or Facebook.