CERT'S PODCASTS: SECURITY FOR BUSINESS LEADERS: SHOW NOTES

Insider Threat and the Software Development Life Cycle

Key Message: Significant insider threat vulnerabilities can be introduced (and mitigated) during all phases of the software development life cycle.

Executive Summary

While most organizations are becoming aware of insider threats to operational systems, it turns out that vulnerabilities can be accidentally and intentionally introduced throughout the development life cycle – during requirements definition, design, implementation, deployment, and maintenance. Once business leaders are aware of these, they can implement practices that will aid in mitigating these vulnerabilities.

In this podcast, Dawn Cappelli, leader of CERT's research efforts on insider threat, discusses lessons learned from real life incidents that exploited vulnerabilities introduced during the software development life cycle and corrective actions to consider.

PART 1: AN UPDATE ON CERT'S INSIDER THREAT RESEARCH

Current Focus

Research results discussed in the first insider threat podcast were based on 150 cases occurring from 1996 to 2002.

Current efforts focus on approximately 100 new cases from 2002 to the present.

Analyzing these cases will assist in updating the current Common Sense Guide to Prevention and Detection of Insider Threats.

Two new models are forthcoming that will address theft of confidential or sensitive information, and fraud. These will complement the current model on insider IT sabotage discussed in the first podcast.

The research team is going to develop an insider threat risk assessment to assist organizations in determining their current exposures.

Linking to the Software Development Life Cycle (SDLC)

Based on an inquiry, the team investigated whether past cases had any root causes in specific phases of the SDLC. What they discovered is that impacts and losses caused by this type of insider threat included:

- Companies going out of business
- Fraud (one case resulted in a loss of $691M)
- Service disruption (telecommunications)
- Modification of critical information (court records, credit records)
- Deliberate virus infection

Several of these cases are described in more detail below.

PART 2: REQUIREMENTS, DESIGN, & IMPLEMENTATION: OVERSIGHTS & FLAWS

Flaws Introduced During Requirements Definition

Developers often forget to think about security. Security requirements are often overlooked such as those for:

- Authentication
- Role-based access controls
- Separation of duties
- Automated data integrity checks

A Case Example

One case involved a state police communications officer who discovered that when looking up drivers' license information, she could also modify this information and create new, illegitimate driver's licenses for people who could not get one. She ended up creating 195 illegal driver's licenses.

The lack of enforced access controls, separation of duties, and data integrity checks during requirements definition contributed to this incident.

Developers need to consider that users may abuse their access, particularly where access to confidential information is involved. Useful steps for detecting this type of breach include tracking which people are accessing which sensitive data, and then generating and reviewing audit reports when data is modified.

Omissions During Design

If a security action was forgotten or overlooked, the team decided to tag this as a requirements flaw. If a requirement was included but not designed correctly, this was designated as a design flaw.

An example is automated work flow processes where designers did not give sufficient attention to security features, such as verifying user authentication information.

Exception handling is another vulnerable area that can be exploited to allow users to circumvent system overrides.

A Case Example

One case involved a loss of $70,000 by a state agency that distributed food stamps. Two insiders discovered that they could circumvent separation of duties when processing a request for expedited food stamps. Such a request did not require supervisor approval or a Social Security number check. Once the food stamps were issued, the request was then marked as denied so it wouldn't show up in any caseworker reports. As far as the system was concerned, no food stamps were issued.

An Implementation Case Example

Neglecting to conduct code reviews can lead to missing insider insertion of backdoors into source code. These can then be used to gain access to the code at a later time without detection.

A web application developer installed backdoors in his software. He was later terminated and used the backdoor to gain access. He sent malicious email to the organization's customers, he altered files and applications, and he initiated a denial of service attack. He did so much damage that the organization ended up declaring bankruptcy as a result.

PART 3: DEPLOYMENT & MAINTENANCE: OVERSIGHTS & FLAWS

Case Examples During Deployment

One case involved a new system that was released into production. The password file that was used in production was the same file that was used in development. So developers were able to access the production system, violating separation of duties.

An insider was responsible for deploying new releases to customer systems. He was able to plant viruses on all customer systems. No two person rule was enforced for making changes.

Oversights During Maintenance

Malicious code is often planted during maintenance. Organizations will often implement effective practices during earlier life cycle phases and then drop the ball during maintenance.

The absence of stringent, enforced change controls can cause many problems. Having change logs is step one but someone must review them on a regular basis to detect suspicious or unauthorized changes.

Case Examples During Maintenance

An insider inserted malicious code that enabled him to commit fraud to the tune of $691M.

A disgruntled employee who was passed over for a promotion planted malicious code into his organization's telecommunications source code with a time delay. Six months later, he left for a new job but before departing he set the malicious code to go off in 6 months. When it went off (12 months after he planted it), it disrupted his former firm's telecommunications functions.

PART 4: SOFTWARE PROCESS & RAISING AWARENESS

Challenges in Tackling the Issues

It's easy to say "do code reviews" and "do change management" but these can be difficult to justify in a resource-constrained environment, which is almost always the case. If the operational system is working well, this is even a tougher sell.

Useful Steps to Take

If an organization is using solid software development processes (such as those described in the SEI's CMMI model) and if developers are aware of the potential for insider threat, they are in a better position to mitigate these risks.

Addressing vulnerabilities earlier in the SDLC saves cost when compared with addressing them later in the life cycle.

Examining CERT's research results on insider IT sabotage and the distinct patterns of behavior leading up to these crimes can provide some clues for detecting "high-risk" individuals during the SDLC. Once you've identified such individuals, you can then review their code and configuration controls logs for software they have put into production.

Resources

CERT's Insider Threat web site

Department of Homeland Security Build Security In web site