Security Misconfiguration

- The system could be completely compromised without you knowing it. All your data could be stolen or modified slowly over time.
- Recovery costs could be expensive.
- Security misconfiguration can happen at any level of an application stack, including the platform, web server, application server, framework, and custom code.
- Developers and network administrators need to work together to ensure that the entire stack is configured properly
- Automatic patches!
- Users with their own accounts that may attempt to compromise the system. Also consider insiders wanting to disguise their actions.
- Attacker accesses default accounts, unused pages, unpatched flaws, unprotected files and directories, etc.

Have you performed the proper security hardening across the entire application stack?

1. Do you have a process for keeping all your **<u>software up to date</u>**? This includes the OS, Web/App Server, DBMS, applications, and all code libraries.
2. Is **<u>everything unnecessary disabled, removed, or not installed</u>** (e.g. ports, services, pages, accounts, privileges)?
3. **<u>Are default account passwords changed or disabled</u>**?
4. **<u>Is your error handling set up to prevent stack traces and other overly informative error messages from leaking</u>**?

5. Are the security settings in your development frameworks (e.g., Struts, Spring, ASP.NET) and **libraries understood and configured properly?**

   The primary recommendations are to establish all of the following:

1. A **repeatable hardening process** that makes it fast and easy to deploy another environment that is properly locked down. Development, QA, and production environments should all be configured identically. This **process should be automated** to minimize the effort required to setup a new secure environment.
2. A process **for keeping abreast of and deploying all new software updates and patches in a timely manner to each deployed environment**. This needs to include all code libraries as well, which are frequently overlooked.
3. A strong **application architecture that provides good separation and security between components.**
4. Consider **running scans and doing audits periodically** to help detect future misconfigurations or missing patches.

Example Scenarios

Scenario #1: Your application relies on a powerful framework like Struts or Spring. XSS flaws are found in these framework components you rely on. An update is released to fix these flaws but you don't update your libraries. Until you do, attackers can easily find and exploit these flaws in your app.

Scenario #2: The app server admin console is automatically installed and not removed. Default accounts aren't changed. Attacker discovers the standard admin pages are on your server, logs in with default passwords, and takes over.

Scenario #3: Directory listing is not disabled on your server. Attacker discovers she can simply list directories to find any file. Attacker finds and downloads all your compiled Java classes, which she reverse engineers to get all your custom code. She then finds a serious access control flaw in your application.

Scenario #4: App server configuration allows stack traces to be returned to users, potentially exposing underlying flaws. Attackers love the extra information error messages provide.