- Secure Deployment
    - Development to production
        - Your programming team has been working on a new application
            - How will you deploy it safely and reliably
    - Patch tuesday
        - Test and deploy Wed? Thurs? Fri?
    - Manage the process
        - Safely move from a non production phase to a full production
- Sandboxing
    - Isolated testing environment
        - No connection to the real world or production system
        - A technological safe space
    - Use during the development process
        - Try some code, break some code, nobody gets hurt
- Incremental development
    - Helps build the application
- Building the application
    - Development
        - Secure environment
        - Writing code
        - Developers test in their sandboxes
- Test
    - Still in the dev stage
    - All of the pieces are put together
    - Does it work?
    - Functional tests
- Verifying the application
    - Quality Assurance (QA)
        - Verifies features are working as expected
            - Validates new functionality
            - Verifies old errors don't reappear
    - Staging
        - Almost ready to roll it out
        - Works feel exactly like the production environment
            - Working with a copy of production data
            - Run performance tests
            - Test usability and features
- Using the application
    - Production
        - Application is live
        - Rolled out to the user community
    - A challenging step
        - Impacts the users
    - Logistical challenges

        - New servers, new software, restart or interrupt service
- Secure baselines
    - The security of an application environment should be well defined
        - All application instances must follow this baseline
        - Firewall settings, patch levels, OS file versions
        - May require constant updates
    - Integrity measurements check for the secure baseline
        - These should be performed often
        - Check against well documented baselines
        - Failure requires an immediate correction

- Provisioning
    - Deploy an application
        - Web server, Database server, middleware server, user workstation configurations, certificate updates, etc.
    - Application software security
        - Operating system, application]
    - Network security
        - Secure VLAN, internal access, external access
    - Software deployed to workstations
        - Check executables for malicious code, verify security posture of the workstation
- Scalability and elasticity
    - Handle application workload
        - Adapt to dynamic changes
    - Scalability
        - The ability to increase the workload in a given infrastructure
        - Build an application instance that can handle 100,000 transactions per second
    - Elasticity
        - Increase or decrease available resources as the workload changes
        - Deploy multiple application instances to handle 500,000 transactions per second.
- Orchestration
    - Automation is the key to cloud computing
        - Services appear and disappear automatically, or at the push of a button
    - Entire application instances can be instantly provisioned
        - All servers, networks, switches, firewalls, and policies
    - Instances can move around the world as needed
        - Follow the sun
    - The security policies should be part of the orchestration
        - As applications are provisioned, the proper security is automatically included
- Deprovisioning

- Dismantling and removing an application instance
    - All good things
- Security deprovisioning is important
    - Don't leave open holes, don't close important ones
- Firewall policies must be reverted
    - If the application is gone, so is the access
- What happens to the data?
    - Don't leave information out there

- Secure coding concepts
    - A balance between time and quality
        - Programming with security in mind is often secondary
    - Testing, testing, and testing
        - The Quality Assurance (QA) process
    - Vulnerabilities will eventually be found
        - And exploited
- Stored procedures
    - SQL database
        - Client sends detailed requests for data
            - -'SELECT' * FROM wp_options WHERE option_id = 1' SQL database query
        - Client requests can be complex
            - And sometimes modified by the user
            - This would not be good
        - Stored procedures limit the client interactions
            - 'CALL get_options'
            - This is that is needed to not allow modifications to the query
        - To be really secure, use only stored procedures
            - The application does not use any SQL queries
- Obfuscation/camouflage
    - Obfuscate
        - Make something normally understandable very difficult to understand
    - Take perfectly readable code and turn it into nonsense
        - The developer keeps the readable code and gives you the chicken scratch
        - Both sets of code perform exactly the same
    - Helps prevent the search for security holes
        - Makes it more difficult to figure out what's happening
        - But not impossible
- Code obfuscation
    - Echo "Hello World"; made into code that's extremely complex and not easy to read
- Code reuse/Dead code
    - Code reuse

        - Use old code to build new applications
        - Copy and paste
    - If the old code has security vulnerabilities, reusing' the code spreads it to other applications
        - You're making this much more difficult for everyone
    - Dead code
        - Calculations are made, code is executed, results are tallied
        - The results are not used anywhere else in the application
    - All code is an opportunity for a security problem
        - Make sure your code is as alive as possible
- Input validation
    - What is the expected input?
        - Validate actual vs expected
    - Document all input methods
        - Forms, Fields, type
    - Check and correct all input (normalization)
        - A zip code should be only X characters long with a letter in X column
            - Fix any data with improper input
        - The fuzzers will find what you missed
            - Dont give them an opening
- Validation points
    - Server side validation
        - All checks occur on the server
        - Helps protect against malicious users
            - Attackers may not even be using your interface
    - Client-side validation
        - The end user app makes the validation decisions
        - Can filter legitimate input from genuine users
        - May provide additional speed to the user
    - Use both
        - But especially server-side validation
- Memory management
    - As a developer, you must be mindful of how memory is used
        - Many opportunities to build vulnerable code
    - Never trust data input
        - Malicious users can attempt to circumvent your code
    - Buffer overflow are a huge security risk
        - Make sure your data matches your buffer sizes
    - Some built-in functions are insecure
        - Use best practices when designing your code
- Third party libraries and SDK's
    - Your programming language does everything
        - Almost
    - Third party libraries and software development kits

        - Extend the functionality of a programming language
    - Security risks
        - Application code written by someone else
        - Might be secure. Might not be secure
        - Extensive testing is required
    - Balancing Act
        - Application features vs unknown code base
    - Data exposure
        - So much sensitive data
            - Credit card numbers, social security numbers, medical info, address, email info
        - How is the application handling the data?
            - No encryption when stored
            - No encryption across the network
            - Displaying info on the screen
        - All input and output processes are important
            - Check them all for data exposure
    - Version control
        - Create a file, make a change, make another change, and another change
            - Track those changes, revert back to a previous version
        - Commonly used in software development
            - But also in operating systems, wiki software, and cloud based file storage
        - Useful for security
            - Compare versions over time
                - Identify modifications to important files
        - A security challenge
            - Historical info can be a security risk

- Exploiting an Application
    - Attackers often exploit application vulnerabilities
        - They find the unlocked door and open it
    - Once you exploit one binary, you can exploit them all
        - The application works the same on all systems
        - A windows 10 exploit affects all Windows 10 users
    - What if all of the computers were running different software?
        - Unique binaries
        - Functionally identical
- Software diversity
    - Alternative compiler paths would result in a different binary each time
        - Each compiled application would be a little bit different
        - But functionality the same
    - An attack against different binaries would only be successful on a fraction of the users

        - An attacker wouldn't know what to exploit to use
        - Make the game much harder to win

- Automation and scripting
    - Plan for change
        - Implement automatically
    - Automated courses of action
        - Many problems can be predicted
        - Have a set of automated responses
    - Continuous monitoring
        - Check for a particular event, and then react
    - Configuration validation
        - Cloud based technologies allow for constant change
        - Automatically validate a configuration before going living
        - Perform ongoing automated checks
- Continuous Integration (CI)
    - Code is constantly written
        - And merged into the central repository many times a day
    - So many chances for security problems
        - Security should be a concern from the beginning
    - Basic set of security checks during development
        - Documented security baselines as the bare minimum
    - Large scale security analysis during the testing phase
        - Significant problems will have already been covered
- Continuous delivery/deployment (CD)
    - Continuous delivery
        - Automate the testing process
        - Automate the release process
            - Click a button and deploy the application
    - Continuous deployment
        - Even more automation
            - Automatically deploy to production
            - No human integration or manual checks