Developer Application Security Checklist

# Secrets Management

* Do not hard-code secrets into an application
* Check code before git commits for sensitive information
* Use Key Vault or similar dedicated key/secret stores where possible

# Authentication

* Use standard practices for handling login, logout, and reset password scenarios
* Look up the gold-standard of current algorithms for hashing passwords, etc.
* Never implement hardcoded service accounts or backdoors
* Force users to use strong passwords
* Always use an account with the least privilege possible to run applications
* Verify authentication even when using REST APIs
* Use public-key authentication instead of passwords wherever possible

# Database

* Store the least amount of information possible – more data means more cost and higher disclosure risk
* Ensure GDPR requirements are met by having a list of where all sensitive information is stored; this also helps in recovering from an attack
* Use SQL prepared statements to avoid SQL injection
* Never run a database as root
* Use UUIDs instead of integers as indices to avoid allowing enumerability on resources (by using sequential integers or similar)

# Development Hygiene

* Make sure your development PC is up to date, patched, and secure – attackers can compromise development workstations to inject malicious payloads
* Make sure your build server/farm is up to date for the same reason
* Use open-source component scanners to ensure dependency libraries are up to date and secure
* Disable stack traces and similar when deploying an application
* Never directly write user content into responses or use untrusted input in any server-side logic
* Use server-side whitelist validation to check user input
* Remove identifying traces from headers of daemons as well as applications; an attacker should not be able to specifically fingerprint versions of your application!
* Log with just enough detail to locate and isolate a problem
* Never log sensitive information!
* Write canary checks for strange behavior to indicate a possible attack
* Rotate tokens/passwords/credentials on a regular schedule
* Don’t use “exec”, “proc”, “system”, etc. to be a part of the web application; the application should not be able to execute arbitrary locally-hosted binaries

# Networking

* Segment development, QA, and production resources from each other
* Use network security groups to limit what traffic can flow in and out of networks
* Use TLS wherever possible and use “strict-transport-security” to force the browser to make all requests as HTTPS
* Cookies should be set to “httpOnly” and “secure”
* Cookies should be scoped by path and domain – be specific!
* Lock down Content Security Policies and remove “unsafe-“ statements
* Use X-Frame-Option and X-XSS-Protection headers
* Redirect all HTTP traffic to HTTPS on the server
* Use CSRF tokens in all input forms and/or new “SameSite” protection (replaces CSRF)
* Minimize the number of open ports on any given server
* Segment responsibility among many servers/containers instead of a single server
* Host any backend services in a segmented network away from public-facing services
* Ensure services are locked down by IP or subnet to only accept traffic from your application
* Restrict outgoing traffic wherever possible

# Infrastructure

* Ensure all daemons and software can be quickly and regularly updated without causing [excessive] downtime
* Use infrastructure description tools like Terraform to define “infra as code” and make deployment easily repeatable
* Leverage Intrusion Detection Systems and WAFs capable of deep packet inspection/filtering
* Rate limit requests to avoid Denial of Service attacks
* Use CAPTCHAs on sensitive pages (such as login) to further thwart excessive requests
* Set and enforce sane size limits on what input data is accepted
* Use DDoS mitigation techniques like global caching proxies (CloudFlare, etc.)
* Serve user-created content from a different root domain to avoid users abusing the domain root’s trust
* Don’t permit redirects from anywhere; ensure requests for redirect come from the application itself
* Use SELinux/AppArmor wherever possible

These recommendations are just a subset of the full [OWASP Application Security Verification Standard](https://github.com/OWASP/ASVS/raw/master/4.0/OWASP%20Application%20Security%20Verification%20Standard%204.0-en.pdf) (v4.0 as of this writing, 8/26/2019). Developers should make themselves familiar with this document to consider themselves “adequately educated” on application security.