PUREBPM

Pure BPM Ltd

Information Security Policy
& Procedure Manual

Policy Frameworks

For a secure application, the following at a minimum are required:

- Organisational management which champions security
- A written information security policy properly derived from national standards
- A development methodology with adequate security checkpoints and activities
- Secure release and configuration management processes

The policies and procedures herein are in part derived from ISO 27002(17799) and take standards and controls from COBIT and other respected information security standards (e.g. Sarbanes-Oxley) and subsidiary best practice guides, such as OWASP.

ISO 27002

ISO 27002 is a risk-based Information Security Management framework directly derived from the AS/NZS 4444 and BS 7799 standards. It is an international standard used heavily in most organisations outside the USA.

COBIT

COBIT is a popular risk management framework structured around four domains:

- Planning and organisation
- Acquisition and implementation
- Delivery and support
- Monitoring

Each of the four domains has 13 high level objectives and each high level objective has a number of detailed objectives.

Web Application Security Policy (WASP)

1. Overview

Web application vulnerabilities account for the largest portion of attack vectors outside of malware. It is crucial that any web application be assessed for vulnerabilities and any vulnerabilities be remediated prior to production deployment.

2. Purpose

The purpose of this policy is to define web application security assessments within Pure BPM Ltd. Web application assessments are performed to identify potential or realised weaknesses as a result of inadvertent mis-configuration, weak authentication, insufficient error handling, sensitive information leakage, etc. Discovery and subsequent mitigation of these issues will limit the attack surface of Pure BPM Ltd services available both internally and externally as well as satisfy compliance with any relevant policies in place.

3. Scope

This policy covers all web application security assessments requested by any individual, group or department for the purposes of maintaining the security posture, compliance, risk management, and change control of technologies in use at Pure BPM Ltd.

All web application security assessments will be performed by delegated security personnel either employed or contracted by Pure BPM Ltd. All findings are considered confidential and are to be distributed to persons on a “need to know” basis. Distribution of any findings outside of Pure BPM Ltd is strictly prohibited unless approved by the Directors of the company.

4. Policy

4.1 Security Assessment

Web applications are subject to security assessments based on the following criteria:
- New or Major Application Release – will be subject to a full assessment prior to approval of the change control documentation and/or release into the live environment.
- Third Party or Acquired Web Application – will be subject to full assessment after which it will be bound to policy requirements.
- Point Releases – will be subject to an appropriate assessment level based on the risk of the changes in the application functionality and/or architecture.
- Patch Releases – will be subject to an appropriate assessment level based on the risk of the changes to the application functionality and/or architecture.
- Emergency Releases – An emergency release will be allowed to forgo security assessments and carry the assumed risk until such time that a proper assessment can be carried out. Emergency releases will be designated as such by an appropriate manager who has been delegated this authority.

4.2 Risk Levels

All security issues that are discovered during assessments must be mitigated based upon the following risk levels. The Risk Levels are based on the OWASP Risk Rating Methodology. Remediation validation testing will be required to validate fix and/or mitigation strategies for any discovered issues of Medium risk level or greater:

- High – Any high risk issue must be fixed immediately or other mitigation strategies must be put in place to limit exposure before deployment. Applications with high risk issues are subject to being taken off-line or denied release into the live environment.
- Medium – Medium risk issues should be reviewed to determine what is required to mitigate and scheduled accordingly. Applications with medium risk issues may be taken off-line or denied release into the live environment based on the number of issues and if multiple issues increase the risk to an unacceptable level. Issues should be fixed in a patch/point release unless other mitigation strategies will limit exposure.
- Low – Issue should be reviewed to determine what is required to correct the issue and scheduled accordingly.

4.3 Security Assessment Levels

The following security assessment levels shall be established:

- Full – A full assessment is comprised of tests for all known web application vulnerabilities using both automated and manual tools based on the OWASP Testing Guide. A full assessment will use manual penetration testing techniques to validate discovered vulnerabilities to determine the overall risk of any and all discovered.
- Quick – A quick assessment will consist of a (typically) automated scan of an application for the OWASP Top Ten web application security risks at a minimum.
- Targeted – A targeted assessment is performed to verify vulnerability remediation changes or new application functionality.

Security assessments will be carried out according to the Testing Protocol (See Appendix 5).

5. Policy Compliance

An employee found to have violated this policy may be subject to disciplinary action, up to and including termination of employment.

A violation of this policy by a temporary worker, contractor or vendor may result in the termination of their contract or assignment with Pure BPM Ltd.

Web application assessments are a requirement of the change control process and are required to adhere to this policy unless found to be exempt. All application releases must pass through the change control process.

Any web applications that do not adhere to this policy may be taken offline until such time that a formal assessment can be performed at the discretion of the Managing Director.

Server Security Policy (SSP)

1. Overview

Unsecured and vulnerable servers continue to be a major entry point for malicious threat actors.

2. Purpose

The purpose of this policy is to establish standards for the base configuration of internal server equipment that is owned and/or operated by Pure BPM Ltd. Effective implementation of this policy will minimise unauthorised access to Pure BPM Ltd proprietary information and technology.

3. Scope

All employees, contractors, consultants, temporary and other workers at Pure BPM Ltd must adhere to this policy. This policy applies to server equipment that is owned, operated, or leased by Pure BPM Ltd or registered under a Pure BPM Ltd-owned internal network domain.

This policy specifies requirements for equipment on the internal Pure BPM Ltd network.

4. Policy

For all internal servers deployed at Pure BPM Ltd, the following items must be met:

- Services and applications that will not be used must be disabled where practical.
- Access to services should be logged and/or protected through access-control methods such as a web application firewall, if possible.
- The most recent security patches must be installed on the system as soon as practical, the only exception being when immediate application would interfere with business requirements.
- Trust relationships between systems are a security risk, and their use should be avoided.
- Do not use a trust relationship when some other method of communication is sufficient.
- Always use standard security principles of least required access to perform a function.
- Do not use root when a non-privileged account will do.
- If a methodology for secure channel connection is available and technically feasible, privileged access must be performed over secure channels, (e.g., encrypted network connections using SSH or IPSec).
- Servers should be physically located in an access-controlled environment.
- Servers are specifically prohibited from operating from uncontrolled cubicle areas.
- All security-related events on critical or sensitive systems must be logged and audit trails saved as follows:
  - All security related logs will be kept online for a minimum of 1 week.
  - Daily incremental tape backups will be retained for at least 1 month.
  - Weekly full tape backups of logs will be retained for at least 1 month.
  - Monthly full backups will be retained for a minimum of 2 years.
- Security-related events will be reported to the Managing Director, who will review logs and report incidents to Directors. Corrective measures will be prescribed as needed. Security-related events include, but are not limited to:

- Port-scan attacks
- Evidence of unauthorised access to privileged accounts
- Anomalous occurrences that are not related to specific applications on the host.

5. Policy Compliance

An employee found to have violated this policy may be subject to disciplinary action, up to and including termination of employment.

A violation of this policy by a temporary worker, contractor or vendor may result in the termination of their contract or assignment with Pure BPM Ltd.

Any program code or application that is found to violate this policy must be remediated within a 3 day period.

Database Credentials Coding Policy (DCCP)

1. Overview

Database authentication credentials are a necessary part of authorising applications to connect to databases. However, incorrect use, storage and transmission of such credentials could lead to compromise of very sensitive assets and be a springboard to wider compromise within the organisation.

2. Purpose

This policy states the requirements for securely storing and retrieving database usernames and passwords (”database credentials”) for use by an application that will access a database running on one of Pure BPM Ltd's networks.

Software applications running on Pure BPM Ltd's networks may require access to one or more database servers. In order to access these databases, a program must authenticate to the database by presenting acceptable credentials. If the credentials are improperly stored, the credentials may be compromised leading to a compromise of the database.

3. Scope

This policy is directed at all system implementer and/or software engineers who may be coding applications that will access a production database server on the Pure BPM Ltd Network. This policy applies to all software (programs, modules, libraries or APIs that will access a Pure BPM Ltd, multi-user production database. It is recommended that similar requirements be in place for non-production servers since they don’t always use sanitised information.

4. Policy

4.1. General

In order to maintain the security of Pure BPM Ltd's internal databases, access by software programs must be granted only after authentication with credentials. The credentials used for this authentication must not reside in the main, executing body of the program's source code in clear text. Database credentials must not be stored in a location that can be accessed through a web server.

4.2. Specific Requirements

Storage of Data Base User Names and Passwords:

- Database user names and passwords may be stored in a file separate from the executing body of the program's code. This file must not be world readable or writeable.

- Database credentials may reside on the database server. In this case, a hash function number identifying the credentials may be stored in the executing body of the program's code.
- Database credentials may be stored as part of an authentication server (i.e., an entitlement directory), such as an LDAP server used for user authentication. Database authentication may occur on behalf of a program as part of the user authentication process at the authentication server. In this case, there is no need for programmatic use of database credentials.
- Database credentials may not reside in the documents tree of a web server.
- Pass through authentication must not allow access to the database based solely upon a remote user's authentication on the remote host.
- Passwords or pass phrases used to access a database must adhere to the Password Policy.

Retrieval of Database User Names and Passwords:

- If stored in a file that is not source code, then database user names and passwords must be read from the file immediately prior to use. Immediately following database authentication, the memory containing the user name and password must be released or cleared.
- The scope into which database credentials may be stored must be physically separated from the other areas of code, e.g., the credentials must be in a separate source file. The file that contains the credentials must contain no other code but the credentials (i.e., the user name and password) and any functions, routines, or methods that will be used to access the credentials.
- For languages that execute from source code, the credentials' source file must not reside in the same browseable or executable file directory tree in which the executing body of code resides.

Access to Database User Names and Passwords:

- Every program or every collection of programs implementing a single business function must have unique database credentials. Sharing of credentials between programs is not allowed.
- Database passwords used by programs are system-level passwords as defined by the Password Policy.
- Developer groups must have a process in place to ensure that database passwords are controlled and changed in accordance with the Password Policy. This process must include a method for restricting knowledge of database passwords to a need-to-know basis.

5. Policy Compliance

An employee found to have violated this policy may be subject to disciplinary action, up to and including termination of employment.

A violation of this policy by a temporary worker, contractor or vendor may result in the termination of their contract or assignment with Pure BPM Ltd.

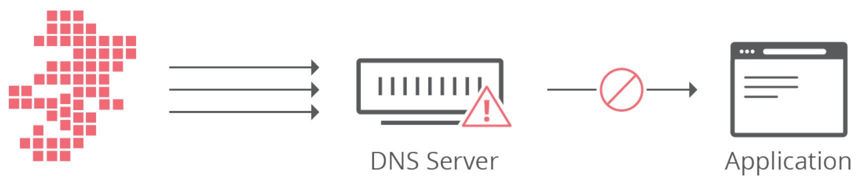
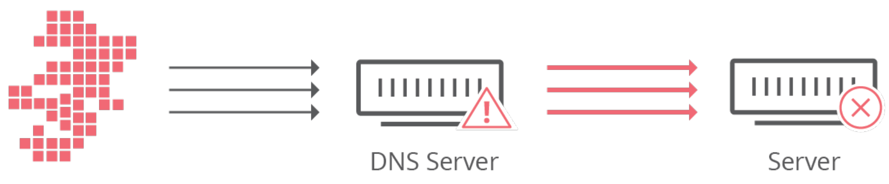
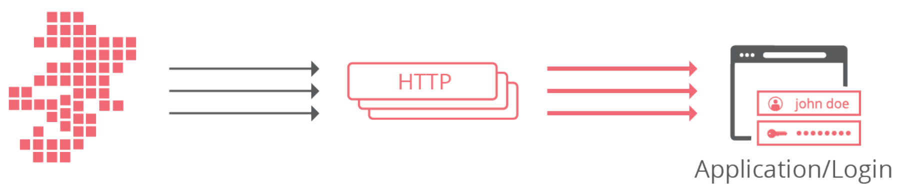
Any program code or application that is found to violate this policy must be remediated within a 3 day period.

Proactive Protection Plan

Denial of Service Attacks

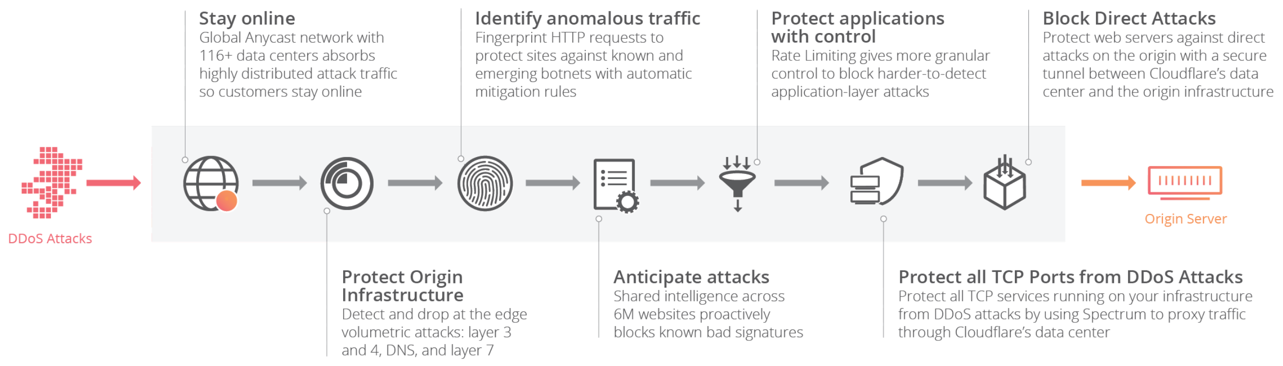
Distributed Denial of Service (DDoS) attacks continue to grow in sophistication and force: more distributed, greater volumes of traffic, and encroaching on the application layer. To combat attacks and stay online a solution that is resilient scalable, and intelligent is required. Current network capacity is 15x bigger than the largest DDoS attack ever recorded. With 30 Tbps of capacity, it can handle any modern distributed attack, including those targeting DNS infrastructure. Shared network intelligence uses an IP reputation database to proactively identify and blocks new and evolving threats.

Common Types of DDoS Attacks

	**DNS Flood** By disrupting DNS resolution, a DNS flood attack will make a website, API, or web application non-performant or completely unavailable.
	**UDP Amplification** An attacker leverages the functionality of open DNS or NTP resolvers to overwhelm a target server or network with amplified request traffic, where the payload size is greater than the size of an originating request.
	**HTTP Flood** HTTP flood attacks generate high volumes of HTTP, GET, or POST requests from multiple sources, targeting the application layer, causing service degradation or unavailability.

Layered Security Defence

A layered security approach combines multiple DDoS mitigation capabilities into one. It prevents disruptions caused by bad traffic, while allowing good traffic through, keeping websites, applications and APIs highly available and performant.



Stay online
Global Anycast network with 116+ data centers absorbs highly distributed attack traffic so customers stay online
Identify anomalous traffic
Fingerprint HTTP requests to protect sites against known and emerging botnets with automatic mitigation rules
Protect applications with control
Rate Limiting gives more granular control to block harder-to-detect application-layer attacks
Block Direct Attacks
Protect web servers against direct attacks on the origin with a secure tunnel between Cloudflare's data center and the origin infrastructure
DDoS Attacks
Origin Server
Protect Origin Infrastructure
Detect and drop at the edge volumetric attacks: layer 3 and 4, DNS, and layer 7
Anticipate attacks
Shared intelligence across 6M websites proactively blocks known bad signatures
Protect all TCP Ports from DDoS Attacks
Protect all TCP services running on your infrastructure from DDoS attacks by using Spectrum to proxy traffic through Cloudflare's data center

Web Application Firewall (WAF)

The enterprise-class web application firewall (WAF) protects applications from common vulnerabilities such as SQL injection attacks, cross-site scripting, and cross-site forgery requests.

OWASP Vulnerability Mitigation

The WAF automatically protects applications from the OWASP top 10 vulnerabilities:

1. Injection
2. Broken Authentication and Session Management
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialisation
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

Protecting Against Zero-Day Vulnerabilities

Security engineers constantly monitor the Internet for new vulnerabilities. When relevant threats are found, WAF rules are automatically applied.

Granular firewall rules are applied in order to stop emerging and sophisticated threats. Rules are based upon multiple request attributes such as user-agent, path, country, query string and IP address.

Rate Limiting

Rate Limiting protects against brute-force login attempts and other types of abusive behaviour targeting the application layer. For example, API usage limits are set to ensure availability and protect against abuse and sensitive customer information is protected against brute force login attacks.

Transport Layer Security (TLS)

All applications are served over HTTPS, encrypting traffic with SSL to ensure nobody can snoop on users' data.

All HTTPS traffic is served over either TLS 1.2 or 1.3, as required by PCI 3.2 compliance due to known vulnerabilities in all earlier versions of TLS and SSL.

TLS Client Auth creates a secure connection between a client and its origin. When a client attempts to establish a connection with its origin server, the device's certificate is validated to check it has authorised access to the endpoint. If the device has a valid client certificate, the device is able to establish a secure connection. If the device's certificate is missing, expired, or invalid, the connection is revoked and a 403 error is returned.

All applications support the HTTP Strict Transport Security (HSTS) protocol that forces clients to use secure connections for every request to the origin server.

Automatic HTTPS Rewrites safely eliminate mixed content issues while enhancing performance and security by rewriting insecure URLs dynamically to their secure counterpart.

Encrypted SNI replaces the plaintext "server_name" extension used in the ClientHello message during TLS negotiation with an "encrypted_server_name." This capability expands on TLS 1.3, increasing the privacy of users by concealing the destination hostname from intermediaries between the visitor and website.

DNSSEC guarantees the applications' traffic is safely routed to the correct servers so that users cannot be intercepted by a hidden "man-in-the-middle" attacker.

Data Breach Prevention

A data compromise can result in the leak of customers' sensitive personally identifiable information (PII) from an application's data store. Attackers often use several attack vectors when attempting to compromise customer data, such as DNS spoofing, snooping of data in transit, brute force login attempts, or malicious payload exploits. For example:

DNS Spoofing

A compromised DNS record, or "poisoned cache", can return a malicious answer from the DNS server, sending an unsuspecting visitor to an attacker's website. This enables attackers to steal user credentials and take ownership of legitimate accounts.

DNSSEC verifies DNS records using cryptographic signatures. By checking the signature associated with a record, DNS resolvers can verify that the requested information comes from its authoritative name server and not a man-in-the-middle attacker.

Snooping of Data In-Transit

Attackers can intercept or “snoop” on unencrypted customer sessions to steal sensitive customer data, including credentials such as passwords or credit-cards numbers.

Fast SSL / TLS encryption and support for the latest security standards enable the secure transmission of sensitive customer data without fear of exposure.

Brute Force Login Attempts

Attackers can wage “dictionary attacks” by automating logins with dumped credentials to brute force their way through a login-protected page.

Rate Limiting detects and blocks hard-to-detect attacks at the network edge, defined by custom rules that set request thresholds, timeout periods, and response codes.

Malicious Payload Exploits

Attackers can exploit application vulnerabilities though malicious payloads. The most common forms include SQL injections, cross-site scripting, and remote file inclusions. Each of these can expose sensitive data by running malicious code on applications.

Web Application Firewall (WAF) rulesets automatically filter out illegitimate traffic targeting the application layer, including GET and POST-based HTTP requests.

Coding Standards

All Developers working on the business’ applications must adhere to the coding standards defined, which have been based upon known best practices and include:

- Architectural guidance
- Minimum documentation levels required
- Mandatory testing and coverage requirements
- Minimum levels of code commenting and the preferred comment style
- Proper use of exception handling
- Correct use of flow of control blocks
- Preferred variable, function, class, and table naming styles
- A preference for maintainable and readable code over clever or complex code

Source Code Control

High performance software engineering requires the use of regular improvements to code, along with associated testing regimes. All code and test changes must be able to be versioned and capable of being reverted.
This is performed by source code revision tools, such as GIT.

Tests must be included with a software revision because tests for later builds will not necessarily match the tests required for earlier builds. So, it is vital that a test is applied to the build for which it was intended.

Testing Protocol

Security is an integral part of the development process. The Software Development Life Cycle (SDLC) must therefore include security tests to ensure security is adequately covered and controls are effective throughout the development process.

Developers should use this guide to ensure that they are producing secure code. These tests should be a part of normal code and unit testing procedures.

Software testers should use this guide to expand the set of test cases they apply to applications. Catching these vulnerabilities early saves considerable time and effort later.

Security specialists should use this guide in combination with other techniques as one way to verify that no security holes have been missed in an application.

Manual Inspections & Reviews

Manual inspections are human-driven reviews that typically test the security implications of the people, policies, and processes, but can include inspection of technology decisions such as architectural designs. They are usually conducted by analysing documentation or performing interviews with the designers or system owners. While the concept of manual inspections and human reviews is simple, they can be among the most powerful and effective techniques available. By asking someone how something works and why it was implemented in a specific way, it allows the tester to quickly determine if any security concerns are likely to be evident. Manual inspections and reviews are one of the few ways to test the software development life-cycle process itself and to ensure that there is an adequate policy or skill set in place. As with many things in life, when conducting manual inspections and reviews we suggest you adopt a trust-but-verify model. Not everything everyone tells you or shows you will be accurate. Manual reviews are particularly good for testing whether people understand the security process, have been made aware of policy, and have the appropriate skills to design or implement a secure application. Other activities, including manually reviewing the documentation, secure coding policies, security requirements, and architectural designs, should all be accomplished using manual inspections.

Advantages:
- Requires no supporting technology
- Can be applied to a variety of situations
- Flexible
- Promotes teamwork
- Early in the SDLC

Disadvantages:
- Can be time consuming
- Supporting material not always available
- Requires significant human thought and skill to be effective

Threat Modelling

Threat modelling has become a popular technique to help system designers think about the security threats that their systems/applications might face. Therefore, threat modelling can be seen as risk assessment for applications. In fact, it enables the designer to develop mitigation strategies for potential vulnerabilities and helps them focus their inevitably limited resources and attention on the parts of the system that most require it. It is recommended that all applications have a threat model developed and documented. Threat models should be created as early as possible in the SDLC, and should be revisited as the application evolves and development progresses. This approach involves:

- Decomposing the application – understand, through a process of manual inspection, how the application works, its assets, functionality, and connectivity.
- Defining and classifying the assets – classify the assets into tangible and intangible assets and rank them according to business importance.
- Exploring potential vulnerabilities - whether technical, operational, or management.
- Exploring potential threats – develop a realistic view of potential attack vectors from an attacker's perspective, by using threat scenarios or attack trees.
- Creating mitigation strategies – develop mitigating controls for each of the threats deemed to be realistic.

Advantages:
- Practical attacker's view of the system
- Flexible
- Early in the SDLC

Disadvantages:
- Relatively new technique
- Good threat models don't automatically mean good software

Source Code Review

Source code review is the process of manually checking a web application's source code for security issues. Many serious security vulnerabilities cannot be detected with any other form of analysis or testing. As the popular saying goes "if you want to know what's really going on, go straight to the source." Almost all security experts agree that there is no substitute for actually looking at the code. All the information for identifying security problems is there in the code somewhere. Unlike testing third party closed software such as operating systems, when testing web applications (especially if they have been developed in-house) the source code should be made available for testing purposes. Many unintentional but significant security problems are also extremely difficult to discover with other forms of analysis or testing, such as penetration testing, making source code analysis the technique of choice for technical testing. With the source code, a tester can accurately determine what is happening (or is supposed to be happening) and remove the guess work of black box testing. Examples of issues that are particularly conducive to being found through source code reviews include concurrency problems, flawed business logic, access control problems, and cryptographic weaknesses as well as backdoors, Trojans, Easter eggs, time bombs, logic bombs, and other forms of malicious code. These issues often manifest themselves as the most harmful vulnerabilities in web sites. Source code analysis can also be extremely efficient to find implementation issues such as places where input validation was not performed or when fail open control procedures may be present. But keep in mind that operational procedures need to be reviewed as well, since the source code being deployed might not be the same as the one being analysed therein.

Advantages:
- Completeness and effectiveness
- Accuracy
- Fast (for competent reviewers)

Disadvantages:
- Requires highly skilled security developers
- Can miss issues in compiled libraries
- Cannot detect run-time errors easily
- The source code actually deployed might differ from the one being analysed

Penetration Testing

Penetration testing has been a common technique used to test network security for many years. It is also commonly known as black box testing or ethical hacking. Penetration testing is essentially the “art” of testing a running application remotely, without knowing the inner workings of the application itself, to find security vulnerabilities. Typically, the penetration test team would have access to an application as if they were users. The tester acts like an attacker and attempts to find and exploit vulnerabilities. In many cases the tester will be given a valid account on the system. While penetration testing has proven to be effective in network security, the technique does not naturally translate to applications. When penetration testing is performed on networks and operating systems, the majority of the work is involved in finding and then exploiting known vulnerabilities in specific technologies. As web applications are almost exclusively bespoke, penetration testing in the web application arena is more akin to pure research. Penetration testing tools have been developed that automate the process, but, again, with the nature of web applications their effectiveness is usually poor. Many people today use web application penetration testing as their primary security testing technique. Whilst it certainly has its place in a testing program, we do not believe it should be considered as the primary or only testing technique. Gary McGraw in summed up penetration testing well when he said, “If you fail a penetration test you know you have a very bad problem indeed. If you pass a penetration test you do not know that you don’t have a very bad problem”. However, focused penetration testing (i.e., testing that attempts to exploit known vulnerabilities detected in previous reviews) can be useful in detecting if some specific vulnerabilities are actually fixed in the source code deployed.

Advantages:
- Can be fast (and therefore cheap)
- Requires a relatively lower skill-set than source code review
- Tests the code that is actually being exposed

Disadvantages:
- Too late in the SDLC

- Front impact testing only

Functional Security Requirements

From the perspective of functional security requirements, the applicable standards, policies and regulations drive both the need of a type of security control as well as the control functionality. These requirements are also referred to as “positive requirements”, since they state the expected functionality that can be validated through security tests. Examples of positive requirements are: “the application will lockout the user after six failed logon attempts” or “passwords need to be six min characters, alphanumeric”. The validation of positive requirements consists of asserting the expected functionality and, as such, can be tested by re-creating the testing conditions, and by running the test according to predefined inputs and by asserting the expected outcome as a fail/pass condition.

In order to validate security requirements with security tests, security requirements need to be function driven and highlight the expected functionality (the what) and implicitly the implementation (the how). Examples of high-level security design requirements for authentication can be:

- Protect user credentials and shared secrets in transit and in storage
- Mask any confidential data in display (e.g., passwords, accounts)
- Lock the user account after a certain number of failed login attempts
- Do not show specific validation errors to the user as a result of failed logon
- Only allow passwords that are alphanumeric, include special characters and six characters minimum length, to limit the attack surface
- Allow for password change functionality only to authenticated users by validating the old password, the new password, and the user answer to the challenge question, to prevent brute forcing of a password via password change.
- The password reset form should validate the user’s username and the user’s registered email before sending the temporary password to the user via email. The temporary password issued should be a one time password. A link to the password reset web page will be sent to the user. The password reset web page should validate the user temporary password, the new password, as well as the user answer to the challenge question.

Risk Driven Security Requirements

Security tests need also to be risk driven, that is they need to validate the application for unexpected behaviour. These are also called “negative requirements”, since they specify what the application should not do. Examples of "should not do" (negative) requirements are:

- The application should not allow for the data to be altered or destroyed
- The application should not be compromised or misused for unauthorised financial transactions by a malicious user.

Negative requirements are more difficult to test, because there is no expected behaviour to look for. This might require a threat analyst to come up with unforeseeable input conditions, causes,

and effects. This is where security testing needs to be driven by risk analysis and threat modelling.
The key is to document the threat scenarios and the functionality of the countermeasure as a factor to mitigate a threat. For example, in case of authentication controls, the following security requirements can be documented from the threats and countermeasure perspective:

- Encrypt authentication data in storage and transit to mitigate risk of information disclosure and authentication protocol attacks
- Encrypt passwords using non reversible encryption such as using a digest (e.g., HASH) and a seed to prevent dictionary attacks
- Lock out accounts after reaching a logon failure threshold and enforce password complexity to mitigate risk of brute force password attacks
- Display generic error messages upon validation of credentials to mitigate risk of account harvesting/enumeration
- Mutually authenticate client and server to prevent non-repudiation and Man In the Middle (MiTM) attacks

Developers' Security Testing Workflow

Security testing during the development phase of the SDLC represents the first opportunity for developers to ensure that individual software components that they have developed are security tested before they are integrated with other components and built into the application. Software components might consist of software artefacts such as functions, methods, and classes, as well as application programming interfaces, libraries, and executables. For security testing, developers can rely on the results of the source code analysis to verify statically that the developed source code does not include potential vulnerabilities and is compliant with the secure coding standards. Security unit tests can further verify dynamically (i.e., at run time) that the components function as expected. Before integrating both new and existing code changes in the application build, the results of the static and dynamic analysis should be reviewed and validated. The validation of source code before integration in application builds is usually the responsibility of the senior developer. Such senior developer is also the subject matter expert in software security and his role is to lead the secure code review and make decisions whether to accept the code to be released in the application build or to require further changes and testing. This secure code review workflow can be enforced via formal acceptance as well as a check in a workflow management tool. For example, assuming the typical defect management workflow used for functional bugs, security bugs that have been fixed by a developer can be reported on a defect or change management system. The build master can look at the test results reported by the developers in the tool and grant approvals for checking in the code changes into the application build.

From the developer's perspective, the main objective of security tests is to validate that code is being developed in compliance with secure coding standards requirements. Developers' own coding artefacts such as functions, methods, classes, APIs, and libraries need to be functionally validated before being integrated into the application build.

The security requirements that developers have to follow should be documented in secure coding standards and validated with static and dynamic analysis. As testing activity following a secure code review, unit tests can validate that code changes required by secure code reviews are properly implemented. Secure code reviews and source code analysis through source code analysis tools help developers in identifying security issues in source code as it is developed. By using unit tests and dynamic analysis (e.g., debugging) developers can validate the security functionality of components as well as verify that the countermeasures being developed mitigate any security risks previously identified through threat modelling and source code analysis.

A good practice for developers is to build security test cases as a generic security test suite that is part of the existing unit testing framework. A generic security test suite could be derived from previously defined use and misuse cases to security test functions, methods and classes. A generic security test suite might include security test cases to validate both positive and negative requirements for security controls such as:

- Authentication & Access Control
- Input Validation & Encoding
- Encryption
- User and Session Management
- Error and Exception Handling
- Auditing and Logging

Developers empowered with a source code analysis tool integrated into their IDE, secure coding standards, and a security unit testing framework can assess and verify the security of the software components being developed. Security test cases can be run to identify potential security issues that have root causes in source code: besides input and output validation of parameters entering and exiting the components, these issues include authentication and authorisation checks done by the component, protection of the data within the component, secure exception and error handling, and secure auditing and logging. Unit test frameworks such as Junit, Nunit, CUnit can be adapted to verify security test requirements. In the case of security functional tests, unit level tests can test the functionality of security controls at the software component level, such as functions, methods, or classes. For example, a test case could validate input and output validation (e.g., variable sanitisation) and boundary checks for variables by asserting the expected functionality of the component.

The threat scenarios identified with use and misuse cases, can be used to document the procedures for testing software components. In the case of authentication components, for example, security unit tests can assert the functionality of setting an account lockout as well as the fact that user input parameters cannot be abused to bypass the account lockout (e.g., by setting the account lockout counter to a negative number). At the component level, security unit tests can validate positive assertions as well as negative assertions, such as errors and exception handling. Exceptions should be caught without leaving the system in an insecure state, such as potential denial of service caused by resources not being deallocated (e.g., connection handles not closed within a final statement block), as well as potential elevation of privileges (e.g., higher privileges acquired before the exception is thrown and not re-set to the previous level before exiting the function). Secure error handling can validate potential information disclosure via informative error messages and stack traces.

Source code analysis and unit tests can validate that the code change mitigates the vulnerability exposed by the previously identified coding defect. The results of automated secure code analysis can also be used as automatic check-in gates for version control: software artefacts cannot be checked into the build with high or medium severity coding issues.

Testers’ Security Testing Workflow

After components and code changes are tested by developers and checked in to the application build, the most likely next step in the software development process workflow is to perform tests on the application as a whole entity. This level of testing is usually referred to as integrated test and system level test. When security tests are part of these testing activities, they can be used to validate both the security functionality of the application as a whole, as well as the exposure to application level vulnerabilities. These security tests on the application include both white box testing, such as source code analysis, and black box testing, such as penetration testing. Grey box testing is similar to Black box testing. In a grey box testing we can assume we have some partial knowledge about the session management of our application, and that should help us in understanding whether the logout and timeout functions are properly secured.

The target for the security tests is the complete system that is the artefact that will be potentially attacked and includes both whole source code and the executable. One peculiarity of security testing during this phase is that it is possible for security testers to determine whether vulnerabilities can be exploited and expose the application to real risks. These include common web application vulnerabilities, as well as security issues that have been identified earlier in the SDLC with other activities such as threat modelling, source code analysis, and secure code reviews.

Usually, testing engineers, rather than software developers, perform security tests when the application is in scope for integration system tests. Such testing engineers have security knowledge of web application vulnerabilities, black box and white box security testing techniques, and own the validation of security requirements in this phase. In order to perform such security tests, it is a pre-requisite that security test cases are documented in the security testing guidelines and procedures.

The main objective of integrated system tests is to validate the “defence in depth” concept, that is, that the implementation of security controls provides security at different layers. For example, the lack of input validation when calling a component integrated with the application is often a factor that can be tested with integration testing.

The integration system test environment is also the first environment where testers can simulate real attack scenarios as can be potentially executed by a malicious, external or internal user of the application. Security testing at this level can validate whether vulnerabilities are real and can be exploited by attackers. For example, a potential vulnerability found in source code can be rated as high risk because of the exposure to potential malicious users, as well as because of the potential impact (e.g., access to confidential information). Real attack scenarios can be tested with both manual testing techniques and penetration testing tools. Security tests of this type are also referred to as ethical hacking tests. From the security testing perspective, these are risk driven tests and have the objective to test the application in the operational environment. The target is the application build that is representative of the version of the application being deployed into production.

The next level of security testing after integration system tests is to perform security tests in the user acceptance environment. There are unique advantages to performing security tests in the operational environment. The user acceptance tests environment (UAT) is the one that is most representative of the release configuration, with the exception of the data (e.g., test data is used in place of real data). A characteristic of security testing in UAT is testing for security configuration issues. In some cases these vulnerabilities might represent high risks. For example, the server that hosts the web application might not be configured with minimum privileges, valid SSL certificate and secure configuration, essential services disabled and web root directory not cleaned from test and administration web pages.

Testing Framework

Pre-Development

Before application development has started:

- Test to ensure that there is an adequate SDLC where security is inherent
- Test to ensure that the appropriate policy and standards are in place for the development team
- Develop the metrics and measurement criteria
- Ensure that there are appropriate policies, standards, and documentation in place.

During Definition & Design

Security requirements define how an application works from a security perspective. It is essential that the security requirements be tested. Testing in this case means testing the assumptions that are made in the requirements, and testing to see if there are gaps in the requirements definitions. Ensure that requirements are as unambiguous as possible.

When looking for requirements gaps, consider looking at security mechanisms such as:

- User Management (password reset etc.)
- Authentication
- Authorisation
- Data Confidentiality
- Integrity
- Accountability
- Session Management
- Transport Security
- Tiered System Segregation
- Privacy

Identifying security flaws in the design phase is not only one of the most cost-efficient places to identify flaws, but can be one of the most effective places to make changes.

Undertake a threat modelling exercise. Develop realistic threat scenarios. Analyse the design and architecture to ensure that these threats have been mitigated, accepted by the business, or assigned to a third party, such as an insurance firm. When identified threats have no mitigation strategies, revisit the design and architecture with the systems architect to modify the design.

During Development

Theoretically, development is the implementation of a design. However, in the real world, many design decisions are made during code development. These are often smaller decisions that were either too detailed to be described in the design, or in other cases, issues where no policy or standard guidance was offered. If the design and architecture were not adequate, the developer will be faced with many decisions. If there were insufficient policies and standards, the developer will be faced with even more decisions.

A code walkthrough (a high-level walkthrough of the code where the developers can explain the logic and flow of the implemented code) should be completed. It allows the code review team to obtain a general understanding of the code, and allows the developers to explain why certain things were developed the way they were.

The purpose is not to perform a code review, but to understand at a high level the flow, the layout, and the structure of the code that makes up the application.

Armed with a good understanding of how the code is structured and why certain things were coded the way they were, the tester can now examine the actual code for security defects.
Static code reviews validate the code against a set of checklists, including:

- Business requirements for availability, confidentiality, and integrity.
- OWASP Guide or Top 10 Checklists for technical exposures.
- Specific issues relating to the language or framework in use.
- Any industry specific requirements.

During Deployment

Having tested the requirements, analysed the design, and performed code review, it might be assumed that all issues have been caught. Hopefully, this is the case, but penetration testing the application after it has been deployed provides a last check to ensure that nothing has been missed.

The application penetration test should include the checking of how the infrastructure was deployed and secured. While the application may be secure, a small aspect of the configuration could still be at a default install stage and vulnerable to exploitation.

During Maintenance & Operations

Monthly or quarterly health checks should be performed on both the application and infrastructure to ensure no new security risks have been introduced and that the level of security is still intact.

After every change has been approved and tested in the QA environment and deployed into the production environment, it is vital that, as part of the change management process, the change is checked to ensure that the level of security hasn't been affected by the change.