	White Box	Black Box
Also known as	Full knowledge assessment	Zero knowledge assessment
Assesses the software's	Structure	Behavior
Root Cause identification	Can identify the exact line of code or design issue causing the vulnerability	Can analyze only the symptoms of the problem and not necessarily the cause
Extent of code coverage possible	Greater; the source code is available for analysis	Limited; not all code paths may be analyzed
Number of False positives and false negatives	Less; contextual information is available	High; since normal behavior is unknown, expected behavior can also be falsely identified as anomalous
Logical flaws detection	High; design and architectural documents are available for review	Less; limited to no design and architectural documentation is available for review
Deployment issues identification	Limited; assessment is performed in pre-deployment environments	Greater; assessment can be performed in pre- as well as post-deployment production or production-like simulated environment.

Table 5.1 – **Comparison between White Box and Black Box security testing**

So then, what kind of testing is "best" to assure the reliability, resiliency and recoverability of software? The answer is "it depends". For determining syntactic issues early on in the SDLC, white box testing is appropriate. If the source code is not available and testing needs to be performed with a true hostile user perspective, then black box testing is the choice. In reality however, it is usually a hybrid of the two approaches, also referred to as gray box or translucent box that is performed to validate security protection mechanisms in place. For a comprehensive security assurance assessment, the hybrid gray box approach is recommended, in which white box testing is conducted pre-deployment in development and test environments and black box testing is performed pre- and post- deployment as well in production-like and actual production environments.

Types of Security Testing

Cryptographic Validation Testing

Cryptographic validation includes the following attestation:

Standards Conformance

Confirmation that cryptographic modules conform to prescribed standards such as the Federal Information Processing Standards (FIPS) 140-2 and cryptographic algorithms used are standard algorithms such as AES, RSA, DSA etc. is necessary.

FIPS 140-2 testing is conducted against a defined cryptographic module and provides a suite of conformance tests to four security levels, from the lowest security to the highest security. Knowledge of the details of each security level is beyond the scope of this book but it is advisable that the CSSLP be familiar with the FIPS 140-2 requirements, specifications and testing as published by NIST.

Environment Validation

The computing environment in which cryptographic operations occur must be tested as well. The ISO/IEC 15408 Common Criteria (CC) evaluation can be used for this attestation. CC evaluation assurance levels don't directly map to the FIPS 140-2 security levels and a FIPS 140-2 certificate is usually not acceptable in place of a CC certificate for environment validation.

Data Validation

FIPS 140-2 testing considers data in unvalidated cryptographic systems and environments as data that is not protected at all, i.e., as unprotected cleartext. The protection of sensitive, private and personal data using cryptographic protection should be validated for confidentiality assurance.

Cryptographic Implementation

The way in which the seed values needed for cryptographic algorithms should be checked is so that they are random and not easily guessable. The uniqueness, randomness and strength of identifiers (such as Session ID) can be determined using phase space analysis and resource and time permitting, these tests need to be conducted. White box tests to make sure that cryptographic keys are not hard code inline code in clear text should be conducted. Additionally, key generation, exchange, storage, retrieval, archival and disposal processes must be validated as well. The ability to decrypt cipher text data when keys are cycled must be checked as well.

Scanning

I once asked one of my students, "Why do we need to scan our networks and software for vulnerabilities", and his response, while amusing was profound. He said, "If we don't, someone else would." When there is very limited or no prior knowledge about the software makeup, its internal working or the computing ecosystem in which it operates, black box testing can start by scanning the network as well as the software for vulnerabilities. Network scans are performed with the goal of mapping out the computing ecosystem. Wireless access points and wireless infrastructure scans must also be performed. These scans help determine the devices, fingerprint operating system, identify active services

(daemons), determine open/closed ports, find used protocols and interfaces, detect web server versions, etc. that make up the environment in which the software will run.

The process of determining an operating system version is known as *OS fingerprinting*. OS fingerprinting is possible because each operating system has a unique way it responds to packets that hit the TCP/IP stack. An example of OS fingerprinting using the Nmap (Network Mapper) scanner is illustrated in *Figure 5.6*.

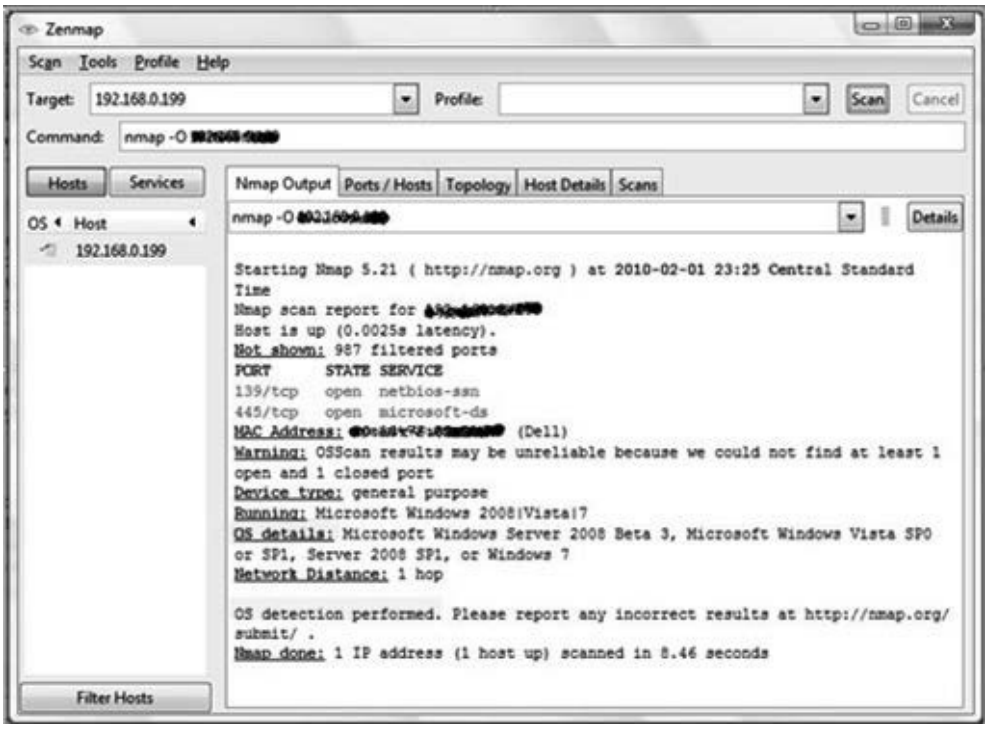

Figure 5.6 – **Fingerprinting Operating System**

There are two main means by which an OS can be fingerprinted – *active* and *passive*. Active OS fingerprinting involves the sending of crafted, abnormal packets to the remote host and analyzing the responses from the remote host. If the remote host network is monitored and protected using intrusion detection systems, active fingerprinting can be detected. The popular Nmap tool uses active fingerprinting to detect OS versions. Unlike active fingerprinting, passive OS fingerprinting does not contact the remote host. It captures traffic originating from a host on the network and analyzes the packets. In passive fingerprinting, the remote host is not aware that it is being fingerprinted. Tools such as Siphon that was developed by the HoneyNet project and P0f use passive fingerprinting to detect OS versions. Active fingerprint is fast and useful when there are large

number of hosts to scan, however it can be detected by IDS and IPS. Passive fingerprinting is relatively slower and best used for single host systems, especially if there is historical data. Passive fingerprinting can also go undetected since there is no active probe of the remote host being fingerprinted.

A scanning technique that can be used to enumerate and determine server versions is known as banner grabbing. Banner grabbing can be used for legitimate purposes such as for inventorying the systems and services on the network, but an attacker can use banner grabbing to identify network hosts that have vulnerable services running on them. The File Transfer Protocol (FTP) port 21, Simple Mail Transfer Protocol (SMTP) port 25 and Hypertext Transfer Protocol (HTTP) port 80 are examples of common ports that are used in banner grabbing. By looking at the Server field in a HTTP response header, upon request, one can determine the web server and its version. This is a very common web server fingerprinting exercise when black box testing web applications. Tools such as Netcat or Telnet are used in banner grabbing. *Figure 5.7* depicts banner grabbing a web server version using Telnet.

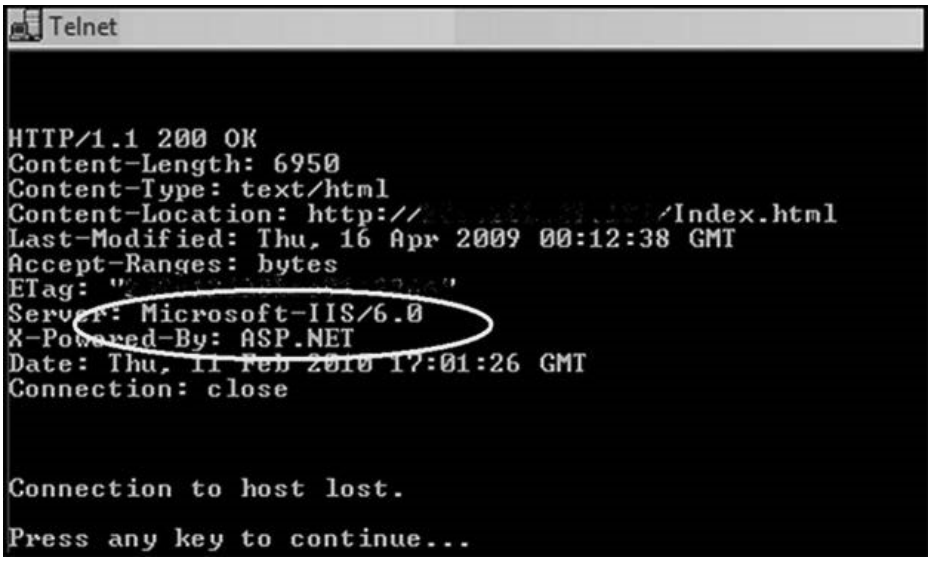

Figure 5.7 – **Banner grabbing web server version**

Scanning can be used to:

- map the computing ecosystems, infrastructural and application interfaces.
- identify server versions, open ports and running services.
- inventory and validate asset management databases.
- identify patch levels.
- prove due diligence due care for compliance reasons.

It should be noted that while many organizations have include scanning as part of their risk management program, the periodicity of performing these scans have usually been on an bi-annual or annual basis which in essence create a false sense of security, even if it meets compliance requirements. The criticality of the software, its content and the data that is processed by the software are the factors that should be used to determine the frequency of the scans. In some cases, daily scans may be necessary.

The three primary types of scanning include: scanning for vulnerabilities, scanning content for threats and scanning for assuring privacy.

Vulnerability Scanning

When scanning is performed with the goal of detecting and identifying security flaws and weaknesses in the software and/or network, it is referred to as vulnerability scanning. The vulnerability scan report that results from this type of scan can be used by development teams, operations teams and management to mitigate identified and confirmed vulnerabilities. Vulnerability scanning can also be used to validate readiness for audit compliance.

Compliance with the PCI DSS requires that organizations must periodically test their security systems and processes by scanning for network, host and application vulnerabilities in the card holder data environment. The scan report should not only describe the type of vulnerability but also provide risk ratings and recommendations on how to fix the vulnerabilities. *Figure 5.8* is an illustration of a sample vulnerability scan report for PCI compliance.

Level	Severity	Description
5	Urgent	Trojan Horses; file read and write exploit; remote command execution
4	Critical	Potential Trojan Horses; file read exploit
3	High	Limited exploit of read; directory browsing; DoS
2	Medium	Sensitive configuration information can be obtained by hackers
1	Low	Information can be obtained by hackers on configuration

Figure 5.8 – **PCI scan report sample**

Most vulnerability scanners use pattern matching (much like a signature-based IDS) against a database of vulnerabilities to detect and identify vulnerabilities. If careful attention is not given to maintain the vulnerability database list with new vulnerability signatures, then the scan report will give a false sense of

security. Additionally, signature-based vulnerability scanners cannot detect new and emerging threats that are not part of the vulnerability database list that is being scanned for.

In addition to network and port scanning, software applications can be scanned as well. Software scanning can be either static or dynamic. Static scanning includes scanning the source code for vulnerabilities; dynamic scanning means that the software is scanned when the software is running i.e., in an operational runtime. Static scanning using source code analyzers is usually performed during the code review process in the development phase, while dynamic scanning is performed using crawlers and spidering tools during the testing phase of the SDLC.

Content Scanning

Advancements in technologies have led to adaptation and shift in the way attackers think about attacking software. The Melissa macro-virus which packed within what seemed to be an innocuous Microsoft Word document a malicious script (macro), malware packed executables that can lead to malicious file execution attacks, image tag injection with HTML content and scripts that can lead to XSS and clickjacking attacks, unsanitized Mashup content and HTML5 tag abuse attacks are some examples of how the content can be used as an attack vector. This is why content scanning is necessary. Content scanning technologies analyze the content within the document (web pages, files, etc.) for malicious content that can exploit unprotected systems. Some content scanners are capable of even analyzing the traffic that is transmitted over secure channels like SSL/TLS and when doing so, they function more or less like a MITM proxy, by capturing encrypted traffic, decrypting it, and analyzing it and re-encrypting it before retransmission. It is important to ensure that content scanners perform deep inspection of both inbound and outbound content.

Privacy Scanning

Privacy scanning is starting to become the norm instead of the exception it used to be a decade ago, due to privacy regulations that mandate the protection of private (personal) information. Privacy scanning helps in detecting potential issues that violate privacy policies and end-user trust.

When the software collects or process private information, it is essential to scan the software to attest the assurance of non-disclosure and privacy. Additionally, when content containing private information is scanned, the scanning technology itself should not violate any end-user privacy requirements or regulations.

Penetration Testing (Pen-Testing)

Vulnerability scanning is passive in nature, meaning we use it to detect the presence of weaknesses and loopholes that can be exploited by an attacker. On the other hand, penetration testing is active in nature, because it goes one step further than vulnerability scanning does. The main objective of penetration testing is to see if the network and software assets can be compromised by exploiting the vulnerabilities that were determined by the scans. The subtle difference between vulnerability scans and penetration testing (commonly referred as pen-testing) is that a vulnerability scan identifies issues that can be attacked, while penetration testing measures the resiliency of the network or software by evaluating real attacks against those vulnerabilities. In penetration testing, attempts to emulate the actions of a potential threat agent (hacker, malware, etc.) are performed. In most cases, pen-testing is done after the software has been deployed, but this need not necessarily be the case. It is advisable to perform black box assessments using penetration testing techniques before deployment for the presence of security controls and after deployment to ensure that they are working effectively to withstand attacks. When penetration testing is performed post deployment, it is important to recognize that "*rules of engagement*" need to be followed and the penetration test itself is methodically conducted. The rules of engagement should explicitly define the scope of the penetration test for the testing team, irrespective of whether they are internal team or an external security service provider. Definition of scope includes restricting IP addresses, the software interfaces that can be tested, etc. Most importantly, the environment, data, infrastructural and application interfaces that are not-in-scope must be identified prior to the test and communicated to the pen-testing team and during the test monitoring must be in place to assure that the pen-testing team does not go beyond the scope of the test. The technical guide to information security testing and assessment, published as Special Publication (SP) 800-115 by the National Institute of Standards and Technology (NIST), provides guidance on conducting penetration testing. The Open Source Security Testing Methodology Manual (OSSTMM) covered in the secure software concepts chapter is known for its prescriptive guidance on the activities that need to be performed before, during and after a penetration test, including the measurement of results. When conducted post-deployment, penetration testing can be used as a mechanism to certify the software (or system) as part of the Validation and Verification (V&V) activity inside of Certification and Accreditation (C&A). V&V and C&A are covered in the software deployment, operations, maintenance and disposal chapter.

Generically the pen-testing process includes the following steps:

1. ***Reconnaissance (Enumeration and Discovery)*** - Enumeration techniques (covered under scanning) such as fingerprinting, banner grabbing, port and services scans, vulnerability scanning, can be used to probe and discover the network layout and the internal workings of the software within that network. WHOIS, ARIN and DNS lookups along with web based reconnaissance are common techniques used for enumerating and discovering network infrastructure configurations.
2. ***Resiliency Attestation (Attack and Exploitation)*** - Upon completion of reconnaissance activities, once potential vulnerabilities are discovered, the next step is to try to exploit those weaknesses. Attacks can be varied ranging from brute forcing of authentication credentials, escalation of privileges to administrator (root) level privileges, deletion of sensitive logs and audit records, disclosure of sensitive information, alteration/destruction of data to causing Denial of Service (DoS) by crashing the software or system.
3. ***Removal of Evidence (Cleanup activities) and Restoration*** - Penetration testers often establish back doors, turn on services, create accounts, elevate themselves to administrator privileges, load scripts, and install agents and tools in target systems. Post attack and exploitation, it is important that any changes that were made in the target system or software for conducting the penetration test are removed and the original state of the system is restored. Not cleaning up and leaving behind accounts, services and tools, and not restoring the system, leaves it with an increased attack surface and any subsequent attempts to exploit the software are made easy for the real attacker. It is therefore essential to not exit from the penetration testing exercise until all cleanup and restoration activities have been completed.
4. ***Reporting and Recommendations*** - The last phase of penetration testing is to report on the findings of the penetration test. This report should include not only technical vulnerabilities covering the network and software, but also include non-compliance with organization policy, and weaknesses in organizational processes and people know-how. Merely identifying, categorizing and reporting vulnerabilities is important, but it adds greater value when the findings of the penetration test result in a plan of

action and milestones (POA&M) and mitigation strategies. The POA&M is also referred to as a management action plan (MAP). Some examples of POA include updating policies and processes, redesigning the software architecture, patching and hardening, defensive coding, user awareness and deployment of security technologies and tools. When choosing a mitigation strategy, it is recommended to compare the POA&M against the operational requirements of the business and balance the functionality expected with the security that needs to be in place and use the computed residual risk to implement them.

The penetration test report has many uses as listed below. It can be used

- to provide insight into the state of security
- as a reference for corrective action
- to define security controls that will mitigate identified vulnerabilities
- to demonstrate due diligence due care processes for compliance
- to enhance SDLC activities such as security risk assessments, C&A and process improvements.

Fuzzing

Fuzzing, which is also known as fuzz testing or fault injection testing, is a brute force type of software testing in which faults (random and pseudo-random input data) are injected into the software and its behavior observed. It is a test whose results are indicative of the extent and effectiveness of input validation. Fuzzing can be used not only to test applications and their programming interfaces (APIs), but also protocols and file-formats. It is used to find coding defects and security bugs that can result in buffer overflows that cause remote code execution, unhandled exceptions and hanging threads that cause DoS, state machine logic faults and buffer boundary checking defects. The data that is used for fuzzing is commonly referred to as fuzz data or fuzzing oracle.

Although fuzzing is a very common methodology of black box testing, not all fuzz tests are necessarily black box tests. Fuzzing can be performed as a white box test or a black box test. In black box fuzzing, the software is sent fuzz data and the symptoms and behavior of the software is analyzed. There is no insight of the internal workings of the software and so there is no guarantee that all actual code paths were covered as part of this type of test. White box fuzzing is sending fuzz data with verification of all code paths. When there is zero knowledge of the software and debugging the software to determine weaknesses is not an option, black box fuzzing is used and when information about the

makeup of the software (like target code paths, configuration, etc.) is known, white box fuzzing is performed.

Based on how the test fuzz data is created, fuzzers can be broadly classified into the following types: Generation-based fuzzers and Mutation-based fuzzers. Fuzz data can either be generated (synthesized) or mutated. The two main techniques in which fuzz data is created is by recursion or replacement. In *recursive* fuzzing, the fuzz data is created by iterating (recursion) through all possible combinations of a set. In *replacive* fuzzing, the fuzz data is created by replacing values from a set of values.

Generation-Based Fuzzing (Smart Fuzzing)

In generation-base fuzzing, the specifications (format) of how the input is expected by the software is programmed into the fuzz tool to create fuzz data by introducing anomalies to the known data content, structures (e.g., checksums, bit flags and offsets), messages and sequencing. In other words, there is foreknowledge of the data format/protocol and the fuzz data is generated from scratch based on the specification/format. This is why generation-based fuzzing is also referred to as smart fuzzing or intelligent fuzzing.

A majority of successful fuzzers operate as generation-based fuzzer and is preferred because they have a detailed understanding of the format or protocol specifications that is being tested. Generation-based fuzzer have relatively greater code coverage is more thorough in its testing approach, but it can be time consuming as the fuzzer has to first import the known data format or structure and then generate variations based on those. This is why appropriate amount of time should be allocated in the project plan when smart fuzzing is part of the test strategy. The main shortcoming of this fuzzing method is fuzzing is based on known formats and structures and so test coverage for new or proprietary protocols is limited or non-existent.

Mutation-Based Fuzzing (Dumb Fuzzing)

Unlike generation-based fuzzing, in mutation-based fuzzing, there is no foreknowledge of the data format or protocol specifications and so the fuzz data is created by corrupting (mutating) existing data samples (if they exist) by recursion or replacement. This is done randomly and blindly and so mutated fuzzing is also referred to as dumb fuzzing. This can be dangerous leading to denial of service, destruction and complete disruption of the software's operations, and so it is recommended to perform dumb fuzzing in a simulated environment as opposed to the production environment.