J Comput Virol Hack Tech (2015) 11:27–49
DOI 10.1007/s11416-014-0231-x

ORIGINAL PAPER

An overview of vulnerability assessment and penetration testing techniques

Sugandh Shah · B. M. Mehtre

Received: 14 November 2013 / Accepted: 10 November 2014 / Published online: 28 November 2014

Abstract All Internet facing systems and applications carry security risks. Security professionals across the globe generally address these security risks by *Vulnerability Assessment and Penetration Testing* (VAPT). The VAPT is an offensive way of defending the cyber assets of an organization. It consists of two major parts, namely *Vulnerability Assessment* (VA) and *Penetration Testing* (PT). Vulnerability assessment, includes the use of various automated tools and manual testing techniques to determine the security posture of the target system. In this step all the breach points and loopholes are found. These breach points/loopholes if found by an attacker can lead to heavy data loss and fraudulent intrusion activities. In Penetration testing the tester simulates the activities of a malicious attacker who tries to exploit the vulnerabilities of the target system. In this step the identified set of vulnerabilities in VA is used as input vector. This process of VAPT helps in assessing the effectiveness of the security measures that are present on the target system. In this paper we have described the entire process of VAPT, along with all the methodologies, models and standards. A shortlisted set of efficient and popular open source/free tools which are useful in conducting VAPT and the required list of precautions is given. A case study of a VAPT test conducted on a bank system using the shortlisted tools is also discussed.

S. Shah
School of Computer and Information Sciences,
University of Hyderabad, Hyderabad, India

S. Shah (✉) · B. M. Mehtre
Center for Information Assurance and Management, Institute for Development and Research in Banking Technology Established by Reserve Bank of India, Hyderabad, India
e-mail: sugandhshah@gmail.com

B. M. Mehtre
e-mail: bmmehtre@idrbt.ac.in

1 Introduction

Security remains one of the major concerns of information systems. The growing connectivity of computers through Internet, the increasing extensibility and the unbridled growth of the size and complexity of systems have made system security a bigger problem now than in the past. Furthermore it is a business imperative to adequately protect an organization's cyber assets by following a comprehensive, and structured approach to provide protection from the risks an organization might face [2]. In an attempt to address the security issues and comply with the mandated security regulations, security experts have developed various security assurance methods including vulnerability detection, proof of correctness, layered design and penetration testing.

Vulnerability Analysis involves discovering a subset of input space with which a malicious user can exploit logical errors in a system to drive it into an insecure state [7]. The overall process of *Vulnerability Detection and Analysis* aims at scanning the target system against all possible test cases and enlisting a rich set of existing loopholes and weak points in the target system. It involves an active analysis of the system for any potential vulnerability, including poor or improper systems configuration, software flaws and other operational weaknesses in the process or technical countermeasures. The vulnerability assessment process is a must to be followed by *Penetration Testing* which is a comprehensive method to test the complete, integrated, operational, and trusted computing base.

Penetration testing determines the difficulty for someone (basically an attacker/hacker) to penetrate an organization's security controls against unauthorized access to its information and information systems. Penetration Testing is done by simulating an unauthorized user attacking the system using

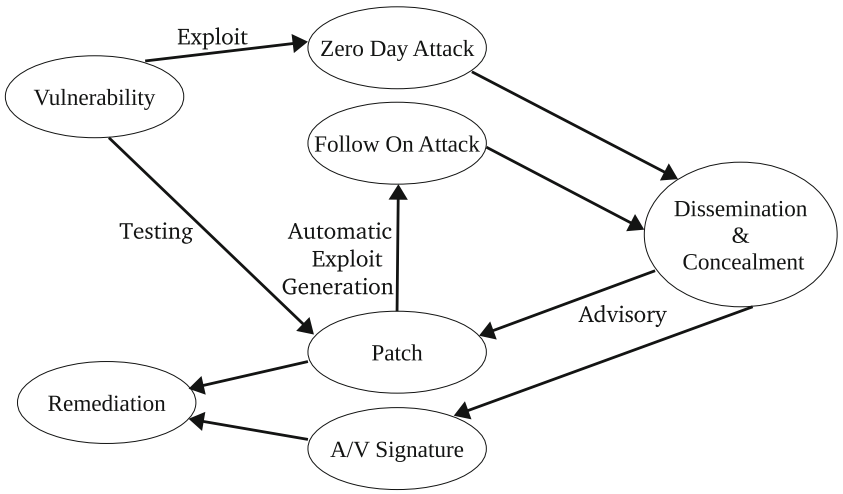

Fig. 1 Life cycle of a vulnerability

either automated tools or manual excellence or a combination of both.

For instance we can say that whenever an organization installs new systems into its technological infrastructure, it conducts a vulnerability assessment test on the newly installed systems with an aim to discover the existing set of weakness and loopholes, if any.

Once the vulnerability assessment is done, the organization tries to patch the required weak points and tries to eliminate the listed vulnerabilities. Now to test the effectiveness of the patching efforts the organization goes for penetration testing, in which the tester simulates an attacker and tries to exploit the system using various tools, techniques and expertise.

VAPT has become a widely used and integral part of quality assurance techniques for the systems used by various financial organizations particularly *Banks*. In fact, many government agencies and trade groups, such as *OWASP* and *OSSTMM*, accredit penetration testers and sanction penetration testing *Best Practices.* The main goal of VAPT is to identify security vulnerabilities under controlled circumstances so they can be eliminated before hackers/attackers exploit them.

The rest of the paper is organized as follows:

Section 2 explains about various *Terminologies* and Background processes that are performed as a part of VAPT, it also includes comparative analysis of all the methodologies and techniques which gives a better understanding of all the concepts. Further Sect. 3 describes the various phases of VAPT, along with the standards and tools available for conducting an efficient VAPT. Case Study of a sample VAPT test conducted in a *Bank* is given in Sect. 4 which will help the testers in getting a practical view of the whole process. Section 5 gives the Conclusion of the paper.

2 Background

This section describes the basic *Terminologies, Components, Techniques* and *Strategies* of Vulnerability Assessment and Penetration Testing. It also gives various benefits of conducting an efficient VAPT on a system or in an organization periodically.

2.1 Vulnerability

A *Vulnerability* is a software or hardware bug or misconfiguration that a malicious individual can exploit [3]. They are the gateways by which different threats are manifested. The process of scanning the infrastructure and identifying such loopholes and weak points is called as *Vulnerability Discovery.* After successfully enlisting all the existing set of vulnerabilities in a system, we conduct *Vulnerability Assessment* which aims to analyze the obtained set of vulnerabilities. The vulnerabilities are prioritized based on their severity and impact over the system.

Figure 1 shows the *Life Cycle* of a vulnerability and Fig. 2 shows the *Risk* involved in the entire *Life Span of a Vulnerability.* which describes the mapping between the Risk level involved and the various phases in the lifetime of a vulnerability.

In Fig. 1 we can see the state diagram for the life cycle of a vulnerability, which shows the stages and the propagation factors from one stage to another in the entire life span starting from *Vulnerability Introduction* to its *Remediation.*

The phases in *Life of a Vulnerability* represented by *Numerals* in Fig. 2 are as follows:

1. ***Security Vulnerability is Discovered.***
2. ***Discovered Vulnerability is made Public.***

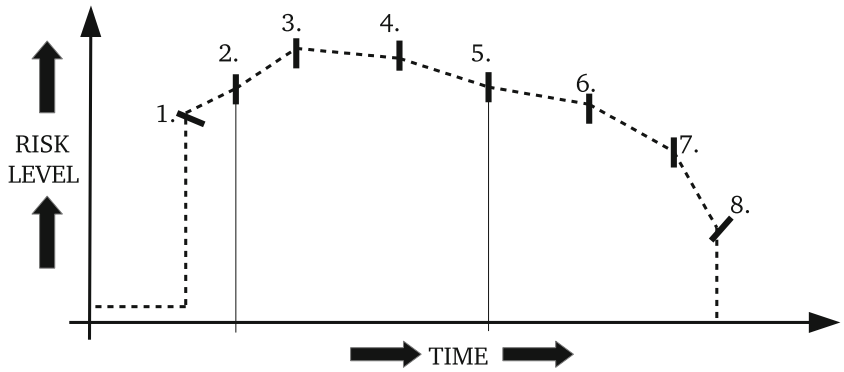

Fig. 2 Risk level vs. phases in life span of a vulnerability (time)

3. ***Vulnerability is Reported to the Vendor.***
4. ***The Vendor Notifies its Clients.***
5. ***Security Tools are Updated (IDS signatures, new modules for VA tools).***
6. ***A Patch is Published.***
7. ***The existence of the Patch gets Widely Known.***
8. ***The Patch is installed in all systems affected.***

From Fig. 2 we can observe that the higher amount of risk is involved between the time when the *Vulnerability gets Discovered (1)* and till the time when the *Vendor acknowledges or notifies its clients (4)*. The Graph attains its peak (Highest Risk) between the time when *The Vulnerability is made Public (2)* and till the time when the *Vendor is Reported (3)*.

A *zero-day* attack is a cyber attack exploiting a vulnerability that has not been disclosed publicly or for which no patch has been deployed till date. A system remains compromised with a zero day attack till the time when the vulnerability gets discovered and a patch is applied to fix the vulnerability.

In our study on vulnerabilities and their life time, we have found some facts as follows:

- Every 11 out of 18 vulnerabilities are not known (zero-day vulnerabilities).
- It has been observed that a zero day attack lasts between 19 days to 30 months, with a median of 8 months and an average of approximately 10 months.
- After vulnerabilities are disclosed (made public), the number of malware variants exploiting them increases 183–85,000 times and the number of attacks on those vulnerabilities increases 2–100,000 times.

Hence we can clearly draw some inferences from these facts like: The public disclosure of vulnerabilities is followed by an increase in the volume of attacks by 5 times. The *Cyber Criminals* (Hackers) keep a close watch on the disclosure of every new vulnerability, in order to start exploiting them.

Detection and *Remediation* of such vulnerabilities involves a huge investment depending on the type of vulnerability and other dependencies. Estimating the cost of a vulnerability is a cumbersome process as it involves the evaluation of multiple factors which in some or the other way influence the existence, detection and exploitation of the vulnerability. Table 1 shows the extract of an article published in *Forbes'* Website which gives the approximate cost of zero-day vulnerabilities related to the products of principal IT security firms.

Table 1 Cost of zero-day vulnerabilities related to products of it firm

NAME OF PRODUCT	COST OF VULNERABILITIES
ADOBE READER	$5,000–$30,000
MAC OS-X	$20,000–$50,000
ANDROID	$30,000–$60,000
FLASH OR JAVA BROWSER PLUG-INS	$40,000–$100,000
MICROSOFT WORD	$50,000–$100,000
WINDOWS	$60,000–$120,000
FIREFOX OR SAFARI	$60,000–$150,000
CHROME OR INTERNET EXPLORER	$80,000–$200,000
APPLE IOS	$100,000–$250,000

Source: www.forbes.com

From Table 1 we can observe the huge cost involved in fixing the vulnerabilities, this huge cost is the result of the aggregated amount incurred on the various influencing factors.

The factors that influence the cost of a vulnerability are as follows:

- Difficulties in identification of vulnerabilities dependent on the security compliance of the application producing company. The longer the time necessary for third parties to discover information, the greater is the cost involved.
- Diffusion level of the concerned application.
- Context of exploited application.
- Applications coming by default with the operating system.
- Necessity of authentication process to exploit the application.
- Typical firewall configuration blocking access to the application.

- Relationship of the vulnerability to server or client application.
- Need of user interaction to exploit the vulnerability.
- Version of software that is affected by the exploit. Recent versions are more costly.
- Dependency of technological context. Introduction of a new technology could in fact lead to less interest in a vulnerability related to an old technology being replaced by the new one.

2.2 Vulnerability types

The major problem faced by security professionals today is the way in which the vulnerabilities are named and grouped. It becomes very confusing if different security professionals and product developers give different names of their choice to the same vulnerability.

To resolve this issue, organizations collectively developed a common language for handling the vulnerabilities, known as CVE and CWE, both of which are sponsored by the *MITRE Corporation* [12]. The CVE—*Common Vulnerabilities and Exposure List,* is a standard in the naming convention of security vulnerabilities making them easier to discuss and document. CWE— *Common Weakness Enumeration,* provides a unified and measurable set of software weaknesses. It enables effective discussion, description, selection and use of software security tools and services that can find these weaknesses in source code and operational systems [12].

For a better understanding and management of software weaknesses and vulnerabilities the Domain of vulnerabilities has been classified into two major parts based on their*Risk* and their *Origin.*

The classification of these security bugs are as follows:

Based on High Risk of the Security Flaws

One way of handling these vulnerabilities is by focusing on the high risk problems. Communities like *OWASP* [8] *and SANS* [13] provide the standard list of most common serious security vulnerabilities. The testers can refer to these ranking lists and handle the vulnerabilities in a system more efficiently. Remediation of vulnerabilities has always been a tough and expensive job, hence it may not be adequate for the organization to resolve all the vulnerabilities at once. In such a scenario the VAPT tester can use these lists and prepare a remediation list with the vulnerabilities of high impact and risk ranked on top.

- ***OWASP Top 10:*** Table 2 hold the list of top 10 vulnerabilities in *Web Application Security* from OWASP [8]. It represents a broad consensus about what the most critical web application security flaws are. The community updates the list every year. The list is generated by different security experts across the globe.

Table 2 OWASP top 10' 2013

RANK	CWE	VULNERABILITY NAME
A1	CWE-929	Injection
A2	CWE-930	Broken Authentication and Session Management
A3	CWE-931	Cross Site Scripting (XSS)
A4	CWE-932	Insecure Direct Object Reference
A5	CWE-933	Security Misconfiguration
A6	CWE-934	Sensitive Data Exposure
A7	CWE-935	Missing Function Level Access Control
A8	CWE-936	Cross Site Request Forgery
A9	CWE-937	Using Components with known Vulnerabilities
A10	CWE-938	Unvalidated Redirects and Forwards

Source: www.owasp.com

- ***CWE/SANS Top 25:*** Unlike *OWASP Top 10* Vulnerability list, which focuses on *Web Applications Security*, Table 3 represents The *SANS Top 25* list which aims at listing the Top 25 vulnerabilities in all kind of applications [13]. The community updates the list on an annual basis with the help of Security experts from SANS and MITRE.

The Complete list is divided into three major parts as follows:

Based On the Origin of the Security Flaws

Another way is to work on the basis of the origin of these security flaws and vulnerabilities, which can be divided into two major groups: *Architecture or Design Flaws* and *Coding or Implementation Bugs.*

- ***Architecture or Design Flaws*** are high-level problems associated with the architecture of the software. These flaws are associated to the SDLC and hence they are comparatively more fundamental and more expensive to change or fix. And they also demand training and help, to understand.
- ***Coding or Implementation Bugs*** are code-level software problems. Unlike design flaws they are not attached to the roots of SDLC and are generally associated to the logical coding part. Hence they are comparatively easier to find through scanning and reviews, these kind of bugs are comparatively smaller and easier to fix. But there tend to be a lot of them [11].

The following *Implementation Bug* and *Design Flaw* descriptions are based on their *Common Weakness Enumeration* descriptions [12].

Some of the frequently observed vulnerabilities which fall under the category of ***Implementation Bugs*** are described as follows.

Cross-Site Scripting (XSS) **(CWE-79)** vulnerabilities occurs when the software does not neutralize or incorrectly

Table 3 CWE/SANS top 25' 2013

DIVISIONS	CWE	VULNERABILITY NAME
1. INSECURE INTERACTION BETWEEN COMPONENTS (This section Deals with Insecure ways of sending and receiving data between components, modules, programs, processes or systems)	CWE-89	SQL Injection
	CWE-78	OS Command Injection
	CWE-79	Cross Site Scripting
	CWE-434	Unrestricted upload of file with dangerous type
	CWE-352	Cross Site Request Forgery
	CWE-601	Open Redirect
2. RISKY RESOURCE MANAGEMENT (This section deals with ways in which software does not properly manage the creation, usage, transfer or destruction of important system resources)	CWE-120	Classic Buffer Overflow
	CWE-22	Path Traversal
	CWE-494	Download of Code without Integrity Check
	CWE-829	Inclusion of functionality from untrusted control sphere
	CWE-676	Use of potentially Dangerous Function
	CWE-131	Incorrect Calculation of Buffer Size
	CWE-134	Uncontrolled Format String
	CWE-190	Integer Overflow or Wraparound
3. POROUS DEFENCES (This section deals with defensive techniques which are often misused, abused or just simply ignored)	CWE-306	Missing Authentication for Critical Function
	CWE-862	Missing Authorization
	CWE-798	Use of Hard Coded Credentials
	CWE-311	Missing Encryption of Sensitive Data
	CWE-807	Reliance on untrusted input in a Security Decision
	CWE-250	Execution with unnecessary Privileges
	CWE-863	Incorrect Authorization
	CWE-732	Incorrect Permission Assignment for critical Resources
	CWE-327	Use of a Broken or Risky Cryptographic Algorithm
	CWE-307	Improper Restriction of Excessive Authentication Attempts
	CWE-759	Use of a One-Way Hash without a Salt

Source: www.sans.org

neutralizes user-controllable input before it is placed in output that is used as a web page and is served to other users. This vulnerability is further classified into three types. *Reflected XSS (Non Persistent)* occurs when the server reads data directly from the HTTP request and reflects it back in the HTTP response. *Stored XSS (Persistent)* occurs when the application stores dangerous data in a database, message forum, visitor log or other trusted data sources. In *DOM-Based XSS,* the client performs the injection of XSS into the page, while in the other types, the server performs the injection.

***SQL INJECTION* (CWE-89)** vulnerabilities occur when the software constructs all or part of an SQL command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.

***Use of Inherently Dangerous Function* (CWE-242)** vulnerability occurs when the program calls a function that can never be guaranteed to work safely. Hence the attacker can easily use common knowledge of their weakness to exploit the application.

***Path Traversal* (CWE-22)** vulnerability occurs when the software uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the software does not

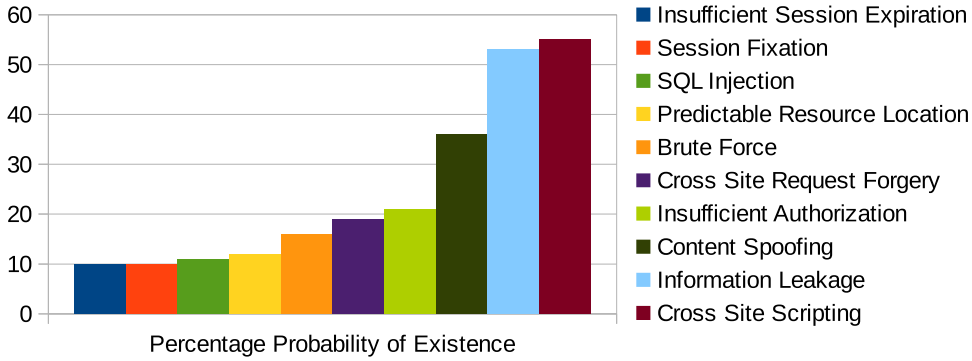

Fig. 3 Probability of different vulnerabilities in a website-2011

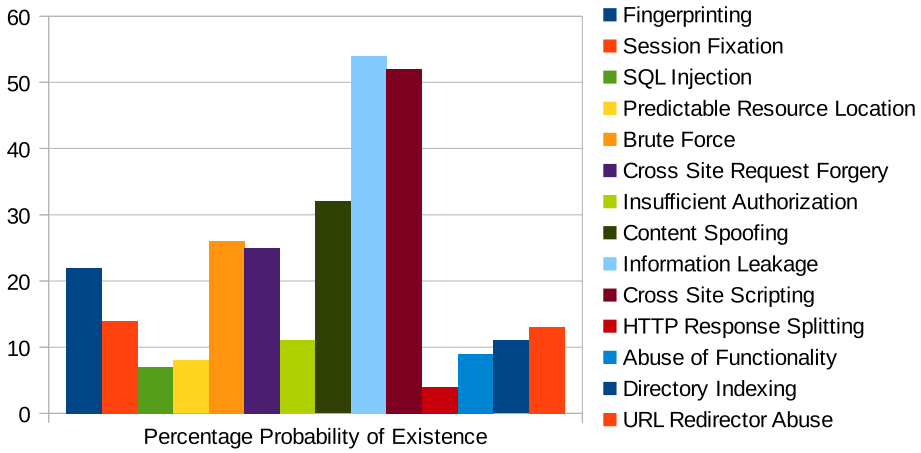

Fig. 4 Probability of different vulnerabilities in a website-2012

properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.

Information Exposure through an Error Message **(CWE-209)** vulnerability occurs when the software generates an error message that includes sensitive information about its environment, user or associated data. The sensitive information may be valuable information on its own (such as a password) or it may be useful for launching other more deadly attacks.

External Control of Assumed Immutable Web-Parameter **(CWE-472)** vulnerability occurs when the web application does not sufficiently verify input that are assumed to be immutable but are actually externally controllable, such as hidden form fields. ***OS Command Injection*** **(CWE-78)** vulnerability occurs when the software constructs a part of an OS command using externally-influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended OS command when it is sent to a downstream component.

Similarly, Some examples of vulnerabilities falling under the category of ***Design Flaws***, are given as follows.

Improper Authorization **(CWE-285)** vulnerability occurs when the software does not perform or incorrectly performs an authorization check when an actor attempts to access a resource or perform an action. When such action control checks are not applied consistently, users are able to access data or perform actions that they should not be allowed to perform.

Insufficient Logging **(CWE-778)** vulnerability occurs when a security-critical event occurs, the software either does not record the event or omits important details about the event when logging it. When such security crucial events are not logged properly, such as a failed login attempt, this can make malicious behavior more difficult to detect and may hinder forensic analysis after an attack succeeds.

Trust Boundary Violation **(CWE-501)** vulnerability occurs when the product mixes trusted and untrusted data in the same data structure or structured message. By combining the trusted and untrusted data in the same data structure, it becomes easier for programmers to mistakenly trust unvalidated data.

Unrestricted Upload of File with Dangerous Type **(CWE-434)** vulnerability occurs when the software allows the attacker to upload or transfer files of dangerous types that can be automatically processed within the product's environment.

To analyze the *Impact* and *Persistence* of these vulnerabilities we analyzed the *Vulnerability Stat Reports* of the year 2011 and 2012 [21]. Figures 3 and 4 represent the percentage

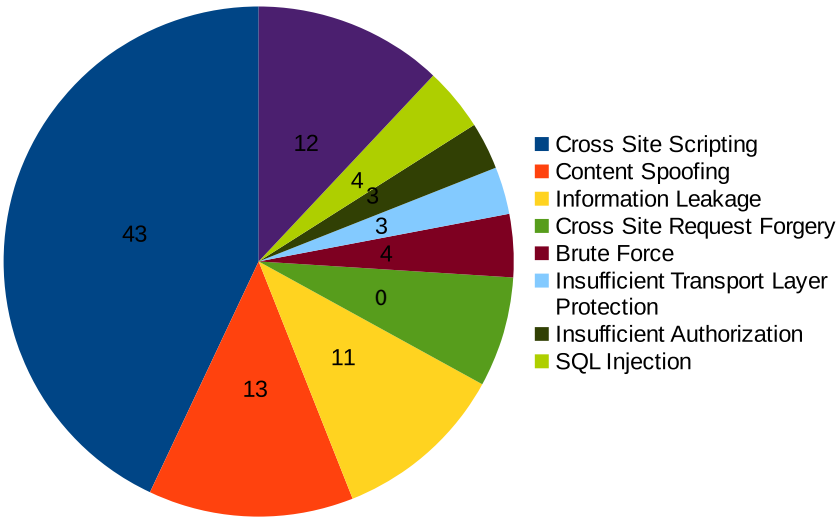

Fig. 5 Overall vulnerability population (%) in 2012

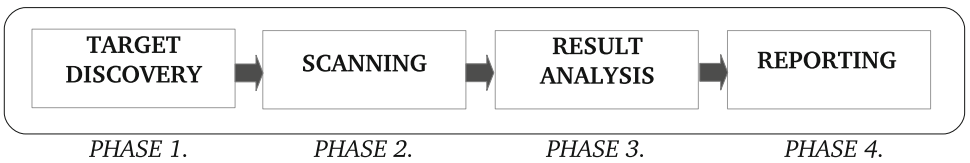

Fig. 6 Phases of vulnerability assessment

likelihood of the existence of at least one serious vulnerability in a website. The percentage figures in the graphs represent the probability of the existence of that specific vulnerability.

Figure 5 shows the overall population of vulnerabilities (in %) for the year 2012 [21]. All these facts and figures give us a complete idea of the prevailing vulnerabilities and their growth. The Organizations can use these details to formulate their *Cyber Defence* strategies.

2.3 Vulnerability assessment

It is the process of vulnerability detection and analysis, This is the first phase of VAPT, in this phase the tester aims at getting familiar with the target system and identifying the list of vulnerabilities associated to the applications and services running on the target system.

As shown in Fig. 6 the process of Vulnerability Assessment has following four phases:

1. Target Discovery
2. Scanning
3. Result Analysis
4. Reporting

Target Discovery: In this phase the tester aims at collecting various crucial informations pertaining to the test target. The VAPT tester tries to gather informations regarding *networks, sub-systems, technologies adopted, system resources, applications, communication infrastructure* etc. These informations collected at this point of time help the tester to generate an image of the target's security infrastructure, which further helps in deciding the run time test strategies and decision making.

Some of the tests conducted in this phase are: *Whois Scanning and Http Header Grabbing.*

The *whois scanning* gives information like:

- Component Name Servers
- Service Registrars
- Server Registration Details
- Contact Details

Similarly *grabbing the Http/Https header* gives informations like:

- Server Version
- Content Type
- Content Length
- X-Powered-By

Scanning: After successfully discovering the target, the testers perform a scan of the complete system with an aim to identify the list of vulnerabilities present on the concerned target. The testers subject the target to various tools and techniques therefore scanning each and every component of the system to find any bug or loophole in the operation of the system, like *unwanted services, open ports, remote connections, weak ciphers, weak passwords* etc. The scanning ends with the list of such potential vulnerabilities in the target which may lead to unwanted data disclosure or loss.

The activities performed in the scanning phase are: *Port Scanning, Network Scanning* and *Web-Application Scanning. Port Scanning* can be done in many ways, Out of all those techniques the *Half TCP Scan* is more appreciated. As the

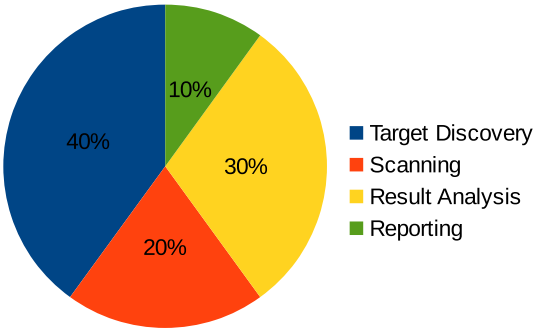

Fig. 7 Phases of VA—effort and time distribution

name suggests, in *Half TCP Scan* the tester sends a *TCP_SYN packet* to the target on the specified port, if the tester receives an *ACK* response the port is considered open, otherwise if a *RST* signal is received the port is considered as closed. If no response is received, the port is timed out. *TCP_SYN* based scanning is more time efficient because it does not complete the full TCP handshaking process, hence the time consumed in sending and receiving of messages is comparatively low.

After obtaining the list of Open ports, the tester initiates the *Network Scanning* and *Web Application Scanning* using automated tools. These tools conduct a complete test on all the network and application resources of the target and generate a report enlisting all the loopholes and mis-configurations detected on all the hosts and other components.

Result Analysis: This phase inherits the output of the Scanning phase and analyses the set of vulnerabilities and other threats identified after scanning. The tester in this phase aims at prioritizing the identified vulnerabilities based on their severity and impact. In practical aspects the results obtained in the scanning phase suffers with a huge amount of *false positives,* these false positives are detected and removed in this phase thereby optimizing the initial list into a more efficient and accurate one. The identified vulnerabilities are ranked and the final prioritized list of vulnerabilities is communicated for further exploitation or remediation.

Reporting: After the successful accomplishment of initial phases, the tester in this phase aims at documenting the various operations performed and results obtained in the entire process of vulnerability assessment. In this phase the tester generates a report out of the work done so far by enlisting the identified set of vulnerabilities along with their severity level and other details. This list can further be used by the organization to call for remediations for the same.

Figure 7 shows the effort and time distribution chart between all the phases of Vulnerability Assessment (VA).

As shown in Fig. 7 the major part of the total effort involved in Vulnerability Assessment goes to *Target Discovery* because most of the processes involved in this phase require Manual Skills and Technological Excellence. In *Scanning* phase due to the availability and use of Automated Tools the effort and time required is less.

After obtaining the Scan results, the process of identifying and testing the existence of vulnerabilities is again a tough and time taking process as the tester in *Result Analysis* phase examines the scan reports to find the vulnerabilities and their impact to prioritize them for mitigation or further exploitation. In the *Reporting* phase the tester simply documents all the test findings for any future reference.

2.4 Vulnerability assessment techniques

Detection, Identification and *Analysis* of vulnerabilities in the target system can be broadly done in following ways:

Manual Testing: as the name suggests this technique propagates without the aid of any automated tools. In this technique the tester is supposed to use his own intellect, experience and expertise to conduct the testing. The Tester subject the target to different test cases and manually observe the response and its variations. If the variation is not found to be in-line with the code-of-conduct, the software is said to be vulnerable.

This *Manual Testing* approach is further classified as:

- ***Exploratory Manual Testing:*** In this technique of manual testing, the tester navigates through the system finding vulnerabilities without any *Test Plan.* The security evaluation in this technique is based on tester's instincts and prior experiences [11].
- ***Systematic Manual Testing:*** In this technique of manual testing, the tester follows a predefined *Test Plan* rather than exploration. The tester in this case makes a thorough study of the system's components and characteristics, based on which he develops an efficient test plan for the concerned system which is followed throughout the process of testing to identify existing vulnerabilities.

The comparative analysis of *Exploratory* and *Systematic Manual Testing* is given in Table 4. The choice of testing strategy should be done wisely based on the *Test Requirements*. Exploratory testing is generally found suitable for testing scenarios where SRS (software requirement specification) is not complete. While systematic testing is preferred in live scenarios where risk of service disruption due to exploration is high.

The major issue with these manual testing approaches was the involvement of huge amount of *Test Time* and *Repetitive Testing Nature* i.e, manual testing consumed a huge amount of test time and also the testers had to perform the same set of testing operations multiple times on different system components in order to detect all the existing loopholes.

Automated Testing: To reduce testing time and further take advantage of the repetitive nature of testing, *Tools* have been

Table 4 Comparison of exploratory and systematic testing

Comparison aspect	Exploratory manual testing	Systematic manual testing
Test Cases	Determined only during Testing	Determined Well in advance
Emphasis	Adaptability and Learning	Prediction and Decision making
Test Controller	Tester's Mind and Experience	Pre Defined Scripts
Testing Nature	Spontaneous	Well Planned
Testing Directions	Obtained from SRS as well as ExploratoryFindings	Obtained from SRS
Motive	Improvement of Test Design	Controlling Tests
Preparation Required	Nil (very Less)	Comparatively very high
Execution Time	High Execution Time	Comparatively Less Time
Risks Involved	High because the Test methods are not Pre-Reviewed	Comparatively Less because all Test Methods are Pre-Reviewed
Cost	Less Cost Effective	Comparatively more Cost Effective

devised to automatically perform many of the same tasks that one does in manual testing [11]. These *Automated Testing Tools* help overcoming all the issues and limitations of manual testing techniques by reducing the human intervention required during the execution of the process.

What happens in automated testing is that the system is subjected to various test cases and a comparison between the systems *Expected* and *Actual outcomes* is performed. If the System's Expectations and Outcomes align, the system is said to be working good. Otherwise the system is suspected to be mis configured or vulnerable.

Static Analysis: In this technique the tester evaluates a system and its components based on its form, structure, content, or documentation, which in any case does not require the program's execution [4]. The technique is different from *Code Review* because the latter is a complete manual process while *Static Analysis* is performed with the help of automated tools. The technique of static analysis is further classified as *Static Analysis for Source Code* and *Static Analysis for Machine Code.* Theoretically the implementation remains the same in both the categories, but practically conducting the latter is more troublesome. The overall process of static analysis aims at examining the codes to identify the weakness in codes which may lead to a vulnerability post execution.

Fuzzing: Also known as *Fuzz Testing,* this technique is often *Automated* or*Semi-Automated.* In this technique the tester sends *Invalid* or *Unexpected* or any *Random Inappropriate Data* as input to the system, and then monitors the system for raised exceptions like *Failures* and *crashes.* Combination of *static fuzzing vectors* are generated to inject as input (random data) into the target system.

Fuzz Testing has been found to be a reliable option for detection of vulnerabilities like : *Buffer Overflow, Cross site scripting, denial of Service* and *SQL Injection.* Apart from these it also plays a major role in the detection of zero day vulnerabilities. While it is not effective in case of threats like *Key Loggers* and *spy wares*.

This Fuzzing technique (Fuzz Testing) is further classified as:

- ***Mutation Based Fuzzing:*** In this technique the input data sample is manipulated blindly without any understanding of its format/structure to create the required *Test data*. Due to this blind approach of the process this technique is also referred as *Dumb Fuzzing. Bit-Flipping* is one of the commonly used *Mutation Based Fuzzing*.
- ***Generation Based Fuzzing:*** Unlike *Mutation Based fuzzing*, In this technique proper understanding of the model, file formats, specifications and protocols of the input data sample is required to generate the required *Test data*. This technique is also referred as *Intelligent Fuzzing.*

To grab a better understanding of the concepts and issues, a comparative analysis of both *Mutation* and *Generation based Fuzzing* techniques is shown in Table 5.

All of these *Vulnerability Discovery* techniques have their own advantages and disadvantages, so the VAPT tester should use his own intellect to choose the type of technique depending on the *Target type* and *Test requirements*.

Table 6 represents a complete list of *Pros* and *Cons* against every vulnerability discovery technique illustrated above. A tester should always be wise and choose the most appropriate testing technique because using an in-appropriate technique can lead to unwanted risks and other disruption threats.

Selection of an effective and appropriate *Discovery Technique* always leads to effective and accurate Test Results.

2.5 Penetration testing

Penetration in this context, can be defined as the *illegitimate acquisition* of *legitimate authority*. A Penetration Testing is

Table 5 Comparison of mutation and generation based fuzzing

Comparison aspect	Mutation based fuzzing	Generation based fuzzing
Test Data	Test Data is generated without any focus on format/structure of input data sample	Test Data is generated using complete understanding of input data format/structure
Execution time	Mutation based fuzzers are found capable of generating large number of Test Cases per second (23 Test Cases per second approx)	Comparatively Generation based fuzzers generate very less number of Test Cases per second (4 Test Cases per second approx)
Vulnerabilities	The number of vulnerabilities identified is comparatively very less due to the lack of strategic Test data generation	Comparatively more number of vulnerabilities are identified due to the use of well formatted and strategic Test data samples
Reliability	Comparatively Less Reliable due to high risk of vulnerabilities going undiscovered	It is more Reliable as it ensures the detection of comparatively larger number of vulnerabilities
Effort Required	Lesser amount of Effort is required as this technique does not demand any special analysis/study of input data for generation of test data	Comparatively more amount of Effort is required due to the need of in-depth analysis of input data vectors for Test data generation
Cost	More Cost Effective as no effort is made in understanding the input	Comparatively Less Cost effective due to expense on analyzing inputs

Table 6 Comparison of all the vulnerability discovery techniques

Discovery technique	Pros	Cons
Manual Testing	• Reduced Short-Term Cost as buying Automated tools is expensive	• Difficult to conduct, the tester is not at ease
	• More likely to identify Real User Issues and Bugs	• Highly Repetitive and not Stimulating
	• Highly Flexible	• Test Templates cannot be Reused
Automated Testing	• High Efficiency with Less Execution Time	• Automated Testing Tools are Highly Expensive
	• Long-Term Cost effective	• Developing of Test Templates and Deploying still takes time
	• Test Templates are Reusable	• Every Testing Tool has its own set of limitations
	• High Availability of Test Results unlike manual testing where test results are seen only by the Tester	• The Tester has almost no control over the test scenario
Static Analysis	• Probability of missing a vulnerability is less	• The Test Results contain large number of False Positives
	• No Limitation on Categories of Vulnerabilities that can be Identified	• Large Execution Time
	• Execution of Source Codes is not required	• Static Code analysis only looks for Patterns that can cause Bugs. So if there is no Pattern matching, it does not means that there is no Bug
	• Flexibility of generating Project Specific Rules	• It is Hard for Static Analyzers to identify non-functional security flaws and logic flaws
	• Bugs can be Identified in the early Development Cycle	
Fuzz Testing	• Can provide Results with very less Effort because User Interaction required is almost Nil	• Fuzz Testers fail to detect Vulnerabilities that do not trigger a full system crash
	• High probability of finding the zero-day Bugs as compared to other technique	• Analyzing the Test Cases generated to crash the system is a hectic task
	• Can judge the Robustness of the entire system Efficiently	• Systems with complex inputs require huge efforts for fuzz generation

a valued assurance assessment tool that benefits both Business and its Operations. A Successful penetration yields the ability to command system facilities to do other than what their owners expected them to do, to gain the full or at least substantial control of a host in a way normally reserved for trusted operation's staff, or to acquire the management interface of an application or the functional equivalent thereof. *Penetration Testing* is the art of finding an open door to penetrate into the target system in an *ethical* approach with an aim to audit and rectify the security infrastructure of the target system.

It helps safeguard the organization against failure/financial loss, preserving corporate image. It identifies and addresses risks even before security breaches actually occur. The Penetration Testing also provides a poof of issue for security investments to senior management.

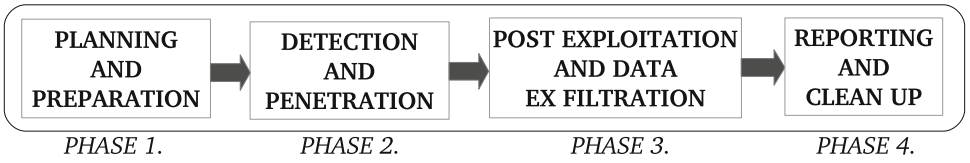

Fig. 8 Phases of penetration testing

As shown in Fig. 8 the complete process of penetration testing is divided into following four phases:

1. Planning and Preparation.
2. Detection and Penetration.
3. Post-Exploitation and Data Ex filtration.
4. Reporting and Clean Up (Destruction of Artifacts).

Planning and Preparation Phase: In this phase the VAPT tester and the organization (Test Target) sit and decide the *Timings, Scope* and *Nature* of testing. All the necessary documents are signed. And further the tester tries to *Investigate, Explore* and *Gather* more and more information about the test target. This *Reconnaissance* strategy propagates in two parts: *Passive Reconnaissance* in which the tester passively gathers all the possible set of details without actually *Touching* the target network i.e, the tester uses external or third party resources to know more about the concerned target. Once the job is done, the attacker enters into *Active Reconnaissance*, in which various experiments are performed over the target to gather out of the line responses of the *Test Target*.

Detection and Penetration Phase: In this phase the attacker tries to compromise the target system in real, by using various tools and techniques to exploit the logical and physical vulnerabilities present in the concerned system. The attacker targets the loopholes in the system and tries to gain access to the system resources by actual exploitation of the identified loopholes. Some of the techniques used in this phase are *Perimeter Penetration, Target acquisition* and *Privilege Escalation.*

Post Exploitation and Data Ex-filtration: In this phase all possible *Ex-Filtration* paths are documented. In this phase the tester identifies things like *Point of Access, Impact on Sensitive Data, Configuration settings* and*Impact on Communication Channels etc.* All these informations are helpful to continue with the exploitation further.

Reporting and Clean Up (Destruction of Artifacts): In this phase the tester primarily aims to deliver a sorted *Report* on all the test findings like what vulnerabilities were identified, the exploitation procedure, proof of concepts, impact of those vulnerabilities and finally the steps required for remediation. The *Destruction of Artifacts* includes removal of any files, tools, exploits, or other test created objects uploaded to the system during testing [18].

Figure 9 shows the effort and time distribution chart between all the above stated phases of *Penetration Testing* (PT). We can clearly observe from the distribution chart that approximately 40 % of the total effort and time is invested in planning the test and gathering information about the target. While 30 % of the effort and time is used in detecting the vulnerabilities and further exploiting them. Rest of the phases include reporting and data ex-filtration which take 10 and 20 % of the resource respectively.

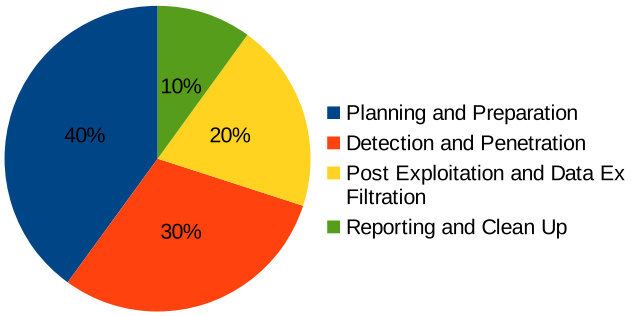

Fig. 9 Penetration testing phases—effort and time distribution

2.6 Penetration testing strategies

For exploitation of the vulnerabilities identified during VA, The VAPT testers need a strategy. So based on the *Type of Auditing* required, there are following three *Penetration Testing Strategies*:

1. Black Box Testing
2. White Box Testing
3. Grey Box Testing

Black Box Testing: In this approach the testers have no prior access to any resources on the test target. They are supposed to figure out all the details along with the loopholes of the system based on their experience and individual expertise. The tester basically aims at auditing the *External* Security boundary of the test target hence the tester simulates the actions and procedures of a real attacker who may be present at some other place outside the boundary of the test target and has no information about the target.

White Box Testing: This approach is contrary to *Black Box*. In this approach the testers are provided with all the necessary informations (about functionalities), access to resources and credentials of the test target, and the tester audits the *Internal* security arrangements of the test target hence the test simulates the actions and procedures of a real internal threat like

Table 7 Comparison of black box, white box and gray box test

Comparison aspect	Black box testing	White box testing	Grey box testing
Target Information	No prior Information about Target or Access to Target Resources	Full Information and Access to Target Resources	Partial Information about target and Partial Access to its Resources
Nature of Testing	User Acceptance Testing	Performed only by Developers and Testers	User Acceptance Testing
Time and Effort	Highly Exhaustive and Time Consuming	Less Exhaustive and Time Consuming	Comparatively Somewhere in Between
Granularity of Test	Low Granularity	High Granularity	Medium Granularity
Fundamentals	Test Design is completely based on External Exceptions as Internal behaviors of system remain unknown	Internal behavior of the system is completely known so Test Design is based on both Internal and External Exceptions	Test Design is made on the basis of database diagrams, data flow diagrams and Internal states of the system
Scope of Testing	Can Test only by Trial and Error method, Only External Boundary can be Tested	Can Test for Data Domain and the Internal Boundaries but not for External Boundaries	Internal and External Boundary, Over flows, Data Domain can all be Tested
Limitations	Not suitable for Testing Algorithms	Suited for all, no specific Limitations	Not suitable for Testing Algorithms

a malicious employee who is present within the boundaries of the test target.

Grey Box Testing: This approach can be understood as the combination of the above two, in this approach the tester is provided with partial disclosure of information and partial access to resources on the test target, and the tester gathers further information by conducting the tests.

For a better understanding of the above mentioned strategies, a comparative analysis of all the three penetration testing strategies was done and the key points are shown in Table 7.

2.7 Types of penetration testing

Penetration Testing is conducted in three major areas *Physical Structure* of the system, *Logical Structure* of the system and the *Response/Work-flow* of the concerned system.

These three areas conclude and define the three types of penetration testing.

1. Network Penetration Testing
2. Application Penetration Testing
3. Social Engineering

Network Penetration Testing: In this technique the tester aims at identifying the security flaws associated with the *Design, Implementation* and *Operation* of the target organization's network. The tester analyses and checks various components like Modems, Remote Access Devices and other connections which may act as an entry point for an attacker to hack into the target's network.

Application Penetration Testing: In this part the tester targets the various applications possessed by the test target primarily the web applications as they remain comparatively more vulnerable to attacks. The tester aims at exposing the effectiveness of an application's security controls by highlighting the risks posed by actual exploitable vulnerabilities. The tester simulates the real attacker and targets the web applications and checks for any vulnerabilities if present which may lead to unauthorized access or Data loss.

Social Engineering: In this part the tester audits the *Workflow* of the target organization, by targeting the human interactions to gather confidential information regarding the target or any of its component systems, which otherwise is supposed to be kept confidential.

Table 8 provides a glimpse of the *Annual Average Vulnerabilities* and their *Remediation Rate* in our leading Industries across the globe as per the records of year 2012 [21].

As per the *Microsoft Records* of historical exploitation trends, it has been observed that vulnerabilities are more often exploited only after a security update is available .

Figures 10 and 11 give an overview of the amount of vulnerabilities that were exploited and later resolved by security updates and the amount of vulnerabilities that were exploited before and after the security updates were available.

Figures 10 and 11 show the *Trends in Software Vulnerability Exploitation* in last few years as per the records of *Microsoft Windows.*

2.8 Benefits of conducting periodic VAPT

As stated earlier, VAPT is a valued assurance assessment tool that benefits both business and its operations. For an

Table 8 Average vulnerability and remediation stats in industries - 2012

Type of industry	Average vulnerable websites (%)	Annual average vulnerabilities	Average remediation rate (%)	Average time to fix vulnerabilities (days)
All	86	56	61	193
Entertainment and Media	91	12	81	33
Financial Services	81	50	67	226
Retail	91	106	54	224
Technology	85	18	61	71
IT	85	114	54	185
Health Care	90	22	53	276
Banking	81	11	54	107
Manufacturing	100	27	55	197
Social Networking	86	20	46	175
Telecommunications	89	20	74	163
Education	100	47	58	342
Energy	100	59	71	144
Insurance	78	39	55	274
Government	100	8	65	48
Non Profit	95	28	41	236
Food & Beverage	100	18	46	36
Gaming	92	17	46	67

Source: www.whitehatsec.com

Organization to remain assured of its *Security Infrastructure,* it must conduct VAPT periodically, it not only assures the security level of its component systems and resources, but also informs about new vulnerabilities and exploits possible, which may lead to financial and data losses. Also from Table 8 we can clearly observe the current vulnerability trend in our leading industries. The *Average Vulnerable Websites* is above 80 % in almost all of the industries, while *Remediation Rate* is less than 60 % in most of them. Also in most of the industries the average time required to fix a vulnerability is above 160 Days, hence to keep the disruption and exploitation risk low one has to discover vulnerabilities as soon as possible by conducting regular VAPT.

VAPT can provide benefits to organizations in both *Business Point of View* as well as *Operational Point of View*.

- ***Business Point of View:*** For any financial organization, its *Corporate Image* remains a big concern, VAPT helps an organization to safeguard against any failure through preventing financial losses, proving due diligence and compliance to industry regulators, customers and shareholders, thereby preserving *Corporate Image* and rationalizing *Information Security investments*. Organizations spend millions of dollars to recover from a security breach due to notification costs, remediation efforts, decreased productivity and lost revenue. The *CSI* study estimates recovery efforts of around $167,713.00 per incident [19]. VAPT being a *Proactive Service* can successfully Identify and address risks before actual security breaches occur, thus preventing any financial loss caused by security breaches. VAPT provides a *Proof of Issue* and a solid case for proposal of investment to senior management, thereby creating high awareness of security's importance at all levels of an organization.
- ***Operational Point of View:*** VAPT helps an organization in shaping *Information Security Strategies* through quick and accurate identification of vulnerabilities, Proactive elimination of identified risks, implementation of corrective measures and enhancement of IT knowledge. VAPT provides detailed information on actual, exploitable security threats if it is encompassed into an organization's security doctrine and processes. By providing the Information required to effectively and efficiently *Isolate* and *Prioritize* vulnerabilities, VAPT can assist the Organizations to *Fine-tune* the test configuration changes or *Patches* to pro-actively eliminate identified risks.

3 VAPT methodology

As illustrated in Sect. 1, The process of VAPT is a *Recursive combination* of two *Different* yet *Interlinked* processes i.e, *Vulnerability Assessment (V.A)* and *Penetration Testing (P.T). Recursive* in the sense that, the process of VAPT must be regularly conducted at specified periodic intervals, because one cannot remain assured of the security of his system for a long time, even after successfully conducting the VAPT. Hence to remain assured of the security and to remain protected from the new exploits, one needs to periodically conduct the VAPT and implement the required patches to remain secured in the near future. The Vulnerability Assessment and Penetration Testing are *Different* yet *Interlinked* processes in the sense that, Vulnerability assessment is designed to yield a prioritized list of vulnerabilities, the customer already knows that they have issues and hence simply needs help identifying and prioritizing them. While the Penetration tests are designed to achieve a specific, attacker-simulated goal and should be requested by customers who are already at their desired security posture. The processes although being different remain interlinked because *Penetration Testing* depends on the outcome of the vulnerability assessment phase i.e, penetration testing can be conducted only after the successfully accomplishing the *Vulnerability Assessment* phase.

VAPT, if done properly, becomes an efficient and cost-effective strategy to protect the organization's systems against attacks. It should never be regarded as a *One-Off-Service,* as doing so will not assure the security arrangements for longer period of time. The test Result of VAPT must be

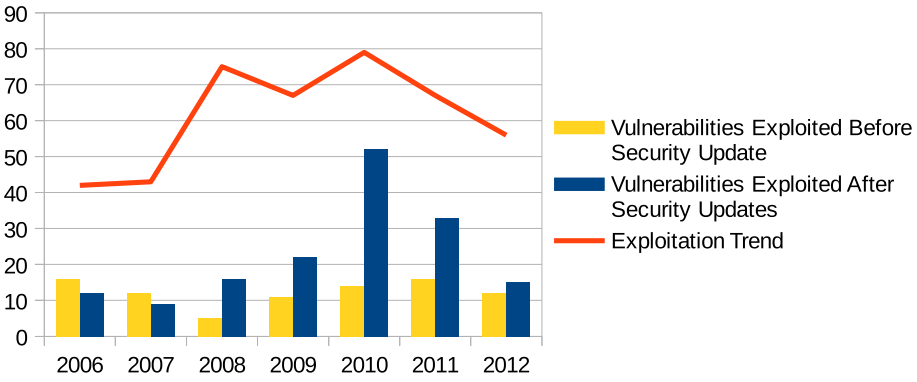

Fig. 10 Trends in vul. exploitation and remediation through updates

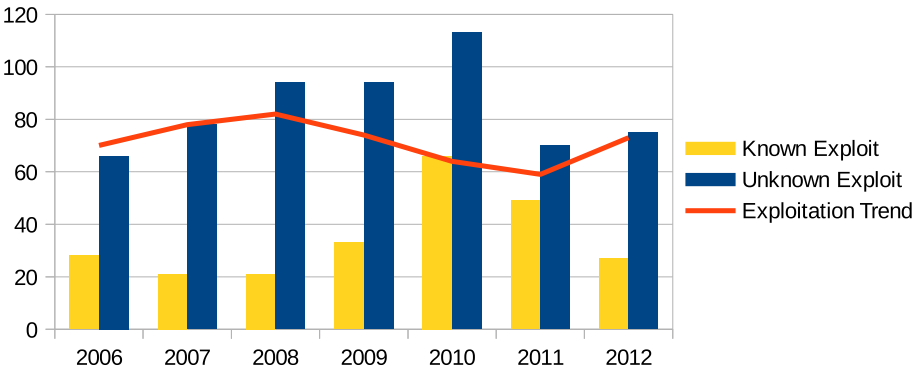

Fig. 11 Trends in vulnerability exploitation before and after update

used to develop effective mitigation plans to resolve the identified issues.

3.1 Phases of vulnerability assessment and penetration testing (VAPT)

As shown in Fig. 12 the process of VAPT is conducted in three major phases:

1. Test Preparation Phase
2. Test Phase
3. Report Generation Phase

Test Preparation Phase: In this phase the tester and the organization meet to decide the *Scope, Objectives, Time* and *Duration* of the test. All the necessary *Documents* regarding the test are organized and finalized. Issues like *Information Leakage* and *Downtime* are resolved and put into legal agreement document.

Some of the major *Documents Required* to conduct VAPT are as follows:

- *Memorandum of Understanding.*
- *Non Disclosure Agreement.*
- *Confidentiality Agreement.*
- *Risk from Jail free Agreement.*

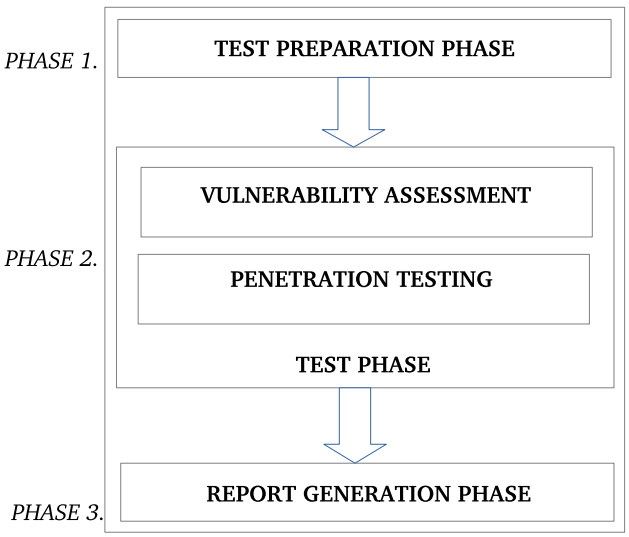

Fig. 12 Phases of VAPT

- *Return of Security Investment Agreement.*
- *Rules of Behavior.*

Test Phase: In this phase the actual testing is done. The tester apply the appropriate techniques for detection and exploitation of vulnerabilities.

The following set operations are performed in this phase.

- ***Information Gathering:*** The success of a VAPT testing is directly proportional to the tester's understanding of the test target. The systematic Information gathering of the target enables the tester to create a complete, efficient and accurate profile of the target's security status. An inefficient information gathering leads to missing key pieces of information, which further leads to an in-accurate VAPT result.
- ***Scanning:*** The sole purpose of scanning is to look for *Holes* in a system's security Armour. The information gathered in previous step is used to perform bulk scanning and probing exercises to further assess the target network space and investigate potential vulnerabilities.
- ***Vulnerability Mapping and Analysis:*** In this part we analyze and map the set of vulnerabilities obtained in the previous step. This part basically aims at prioritizing the vulnerabilities by identifying their severity and impact.
- ***Attack Generation:*** In this part we inherit the prioritized set of vulnerabilities from above step, and based on the available knowledge base and expertise we identify or create exploits, which can be used to target the identified set of vulnerabilities.
- ***Target Exploitation:*** After the identification/Creation of suitable exploits, in this part we deploy those exploits over the marked vulnerabilities in the system with the sole aim to gain access to the system resources.

Report Generation Phase: This phase deals with the thorough investigation and validation of all the test results thereafter *Documentation* and *Reporting* of all the *Test findings* and a *Mitigation Plan.* The report generated at this phase is delivered to the concerned authority of the test target along with the mitigation plan which holds suggestions for remediation of the identified vulnerabilities and exploits, to ensure the security of the system in future.

After generation of the report the VAPT tester tries to bring the system back to its Pre-Test state. He removes all the exploits, configurations and other changes that were made to the system in order to capture the response during VAPT.

Figure 13 Shows the Effort and Time Distribution among the described Phases of VAPT.

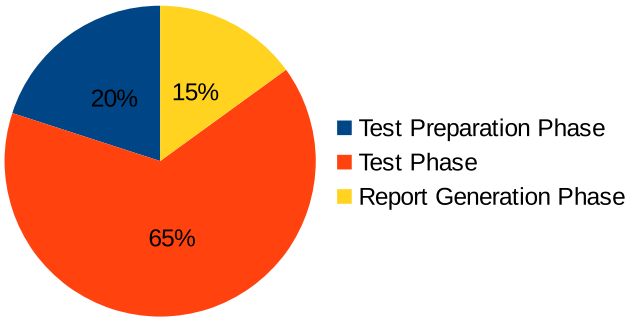

Fig. 13 Phases of VAPT—effort and time distribution

3.2 Vulnerability assessment and penetration testing (VAPT) standards

To ensure the accuracy and effectiveness of the process, the testers follow some standards to conduct penetration testing. These standards ensure the correctness of the procedures being adopted therefore reducing the risk of failure.

There are four major *Penetration testing standards*, adopted across the globe.

1. Open Source Security Testing Methodology Manual (OSSTMM)
2. Payment Card Industry Data Security Standards (PCI-DSS)
3. Open Web Application Security Project (OWASP)
4. International Organization for Standardization (ISO/IEC-27001)

Open Source Security Testing Methodology Manual (OSSTMM): It is a peer-reviewed manual of security testing and analysis which results in verified facts [14]. These facts provide actionable information that can measurably improve your operational security. The OSSTMM basically concentrates on improving the quality of enterprise security as well as the methodology and strategy of testers. Using OSSTMM guidelines the testers have verified information specific to their needs on which they can reliably base their security decisions, hence the testers no longer have to rely on general best practices, anecdotal evidences or superstitions.

The OSSTMM was released by *Pete Herzog* and its standards are maintained by *The Institute for Security and Open Methodologies (ISECOM).*

Payment Card Industry Data Security standards (PCI-DSS): It is a proprietary Information security standard for organizations that handle cardholders informations for the major *Debit, Credit, Prepaid, E-Purse, ATM* and *POS* cards, it provides the requirements and security assessment procedures for the testers when they conduct penetration test primarily in *Banks* and *E-Commerce sites* [15]. The PCI-DSS standards were developed to encourage and enhance cardholders data security and facilitate the broad adoption of consistent data security measures globally. PCI-DSS provides a baseline of technical and operational requirements designed to protect cardholders data. It applies to all entities involved in payment card processing- including merchants, processors, acquires, issuers and service providers, as well as all other entities that store, process or transmit cardholders data. The standard comprises a minimum set of requirements for pro-

tecting cardholders data, and may be enhanced by additional controls and practices to further mitigate the risks.

The PCI-DSS standards are maintained and distributed by *PCI Security standards Council.*

Open Web Application Security Projects (OWASP): The *OWASP Application Security Verification Standard (ASVS)* was first published in December 2008 [16]. The primary aim of this standard is to normalize the range of coverage and level of rigor available in the market when it comes to the issue of performing *Application Level Security verifications* using a *Commercially Workable Open standard* [16]. The standard basically aims at the security issues regarding *Web Applications* and provides a basis for testing application technical security controls, as well as any technical security control in the environment, that are relied on to protect against vulnerabilities such as *Cross Site Scripting* and *SQL Injection.* This standard can be efficiently used by the testers to establish a level of confidence while dealing with the security issues of web applications.

The *OWASP ASVS* is maintained by *OWASP Foundation* and is sponsored by *Aspect Security.*

International Organization for Standardization 27001:2013 (ISO/IEC 27001): It is an Information Security Standard published on 25 September 2013, thereby replacing its previous version ISO/IEC 27001:2005. It is basically the specification for *Information Security Management System (ISMS).* ISO/IEC 27001 defines how to organize *Information Security* in any kind of Organization, profit or non-profit, private or state-owned, small or large [17]. It is safe to say that this standard is the foundation of Information Security Management. It is a standard written by the world's best experts in the field of Information security and aims to provide a methodology for the implementation of Information Security in an Organization. It also enables an Organization to get Certified, which means that an independent certification body has confirmed that Information Security has been implemented in the best possible way in the concerned Organization.

The Standard is Regulated and Maintained by *International Organization for Standardization (ISO)* and *International Elctrotechnical Commission (IEC).*

3.3 Tools to conduct vulnerability assessment and penetration testing (VAPT)

To conduct an efficient and Accurate VAPT, the tester must use the most efficient and best fit set of tools. There are many *Commercial* and *Open Source* tools to satisfy the purpose, but the selection must be made very wisely. As the Commercial tools for VAPT may lead to a huge amount of Investment, hence depending on the *Size, Scope* and *level* of the testing required, the tester must choose the most effective set of tools.

Tables 10, 11, 12 and 13 represent some very efficient and productive set of *Open Source* and*Free tools* which are frequently used by testers across the globe to conduct VAPT. The advantage of using Open Source tools is that the Source Codes as well as executional controls are always available i.e, one can go for a code review and check if the tool being used contains some malicious code. The Tools in these Tables have been shortlisted on the basis of their performance, accuracy and popularity. The Source for each tool has also been given in the table, hence the testers can easily browse the source and download the concerned tool.

3.4 Vulnerability assessment and penetration testing (VAPT) models

A VAPT model is actually a *Blueprint* of the overall process, in which the tester conducts an efficient VAPT. The model prevents the testers from deviating from the right track, thereby helping the testers to analyze their course of work and further protecting them from failure.

The VAPT models are broadly classified into two categories:

1. Flaw Hypothesis Model
2. Attack Tree Model

Flaw Hypothesis Model: This model was developed at *System Development Corporation* and it provides a framework for VAPT studies [25]. It is basically a *System Analysis* and *Penetration Prediction* technique which compiles a list of *Hypothesized Flaws* in the concerned system by analyzing the specification and documentation for the system. These *Hypothesized flaws* are devised on the basis of the tester's experience and expertise on the concerned system type. Once the list of such flaws is devised, the flaws are then prioritized on the basis of the estimated probability that a flaw actually exists, and on the ease of exploiting it to the extent of control or compromise. The final prioritized list is then used to direct the actual testing process.

The Flaw Hypothesis model proceeds in following five steps:

- ***Information Gathering:*** In this part the tester tries to become familiar with the system's functioning, its components and resources. And examines the system's design, implementation, operating procedures and Use.
- ***Flaw Hypothesis:*** Based on the knowledge gained in the first step and with the help of the previous experiences and expertise, the tester hypothesizes flaws in the concerned system. The actual existence and exploitation of these flaws is checked in the next steps.
- ***Flaw Testing:*** In this part the testers perform the actual testing of their list of hypothesized flaws. If a flaw does not

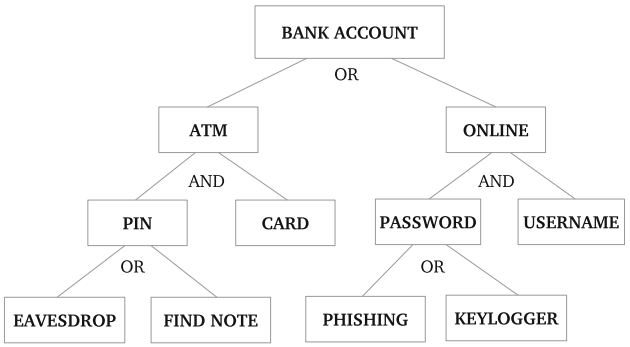

Fig. 14 Sample attack tree model for hacking bank account

exist or cannot be exploited, the testers go back to the previous step. Otherwise if the flaw exists and is exploitable they proceed to the next step.

- ***Flaw Generalization:*** Once the flaw is successfully exploited, the testers try to generalize the concerned vulnerability and try to find others similar to it by feeding their new understanding or hypothesis to second step and keep iterating until the test is complete and there are no more vulnerabilities to be addressed.
- ***Flaw Elimination:*** This is an additional step in which the testers document and report the test findings and try to suggest mitigation plans to resolve the identified vulnerabilities.

Attack Tree Model: This model provides a *Formal* and *Methodological* way of representing the possible *Attacks* against a system in a *Tree Structure,* where all the *Nodes* of the tree represent an Attack or a Specific Goal [25]. The *Root Node* of the tree represents the primary goal of the tester (*Simulated Attacker)* and the *Child Nodes* are the *Refinements* or *Sub Goals* of the goals of their *Parent Nodes*. This *Refinement* can be either *Conjunctive* (aggregation) or *Disjunctive* (Choice).

Suppose there are four *Child Nodes* to a *Parent Node* and if the refinement of the *Child Nodes* is *Conjunctive*, Then all the *Sub Goals* of all the *Child Nodes* are aggregated to obtain the *Parent's Goal*. But in case of a *Disjunctive* refinement, if a *Parent node* has four *Child Nodes*, then there is a choice and any one of the *Child Goals* can lead to the *Parent Goal*.

The following Fig. 14 shows a sample *Attack Tree Model* to hack into a *Bank Account*, where *AND* represents *Conjunctive Refinement* and *OR* represents *Disjunctive Refinement.*

3.5 Precautions while conducting VAPT

To proceed with the *Precautions* we first need to address the various *Risk Factors* involved in the testing process.

Risks Involved in VAPT: *Security Testing* causes risks to the target by its very nature. Like an *Attacker* the *VAPT Tester* deliberately leaves the relatively safe ground of intended use and expected activities. Security Testing is inherently invasive where it employs techniques similar to those used in an attack [9].

The further *Specific Risks* of VAPT can be categorized as follows:

- ***Technical Risks***: These are the risks caused directly by the *Testing Activities* or by the *System being Tested.* Some of the major technical risks are *Failure* of the target or connected systems, *Disruption* of service, *Reduced* Performance, *Modification* or *Contamination* of data, *Disclosure* of confidential data to third parties.
- ***Organizational Risks***: The VAPT testing also involves some organizational side effects like Unnecessary *Triggering* of incident handling processes, *Disruption* of business processes and functions and *Loss of Reputation* if third parties are effected.
- ***Legal Risks***: These kind of risks are encountered due to legal obligations and possible side effects of third parties like *Violation* of *Legal Obligations* and *Inadvertently committing* punishable acts.

Precautions in VAPT: Based on the risk factors involved in VAPT, the testers need to focus on some precautions in order to prevent any unexpected harm to the target system. The testers generally adopt the following major strategies to do so:

- ***Indirect Testing:*** In this technique, the testers instead of testing the actual defects, aim to collect sufficient evidences to conclude that a vulnerability is likely to be present. The technique is useful when dealing with known vulnerabilities.
- ***Limited Exploitation:*** The testers try to prefer *Test Cases* that demonstrate the vulnerability and its exploit, and try to reduce the actual amount of exploitation. The testers use certain *Payloads* that show measurable effects without severe side effects.
- ***Delayed Effects:*** Sometimes, if possible, testers design tests for delayed effects. The testers then evaluate the test results inside the system and cancel or inhibits any further processing before it would occur. The strategy is effective in cases where the tests have real-world effects.
- ***Interruptible Testing:*** In some testing scenarios the testers have to ensure that they can interrupt their testing at any time, so that they can immediately react if any unintended consequences are observed.
- ***Throttled Tools:*** When using automated tools to execute large number of individual tests, the testers must ensure the target's won't be overloaded, as it may result into *Disruption* of services.

- ***Avoiding Lock-outs:*** Sometimes repeated tests might trigger functions designed to *Lock-Out* attackers. For instance password protected systems often limit the number of failed login attempts. Testers must ensure they do not lead to such *Denial of Service* scenarios.
- ***Testing Tests:*** *Exploratory testing* approaches where the testers develop new tests based on their vulnerability hypotheses, is inherently more risky than executing well planned tests. Hence testers must use a lab environment to develop and try tests before deploying them against real targets.
- ***Partial Isolation and Replication:*** Subsystems of the target system can sometimes be reconfigured for testing either dynamically or by setting up a replica for the subsystem in a different configuration. Thereby reducing the side effects of testing.
- ***Rules of Engagement:*** Last but not the least, tester and the client (organization) must establish clear and unequivocal rules of engagement, thereby clearly specifying the targets, test timings and the limits of testing. The third parties that might get affected by the concerned tests must also be notified.

4 Case study

In this section we will discuss about a VAPT test conducted in a*Bank*. Due to confidentiality issues we will not refer the Bank by its actual name, we will use the *Nomenclature* as *Bank A* in the entire study.

Objective of Test: The purpose of this assessment was to identify technical as well as logical vulnerabilities in the web applications of Bank A and provide recommendations for risk mitigation that may arise on successful exploitation of these vulnerabilities.

Project Team: A team of 2 *IT Professionals* from Bank A and a *Security Consultant* from an *External Agency* was finalized to carry out the task.

Project Time-line: The Project Started on 27th March 2013 and Ended on 30th March 2013 the timings for VAPT were fixed between 09:30 h IST to 18:00 h IST.

Approach: The Test was conducted as a *Black Box* followed by *Grey Box* exercise, implying that the testing team was not given any prior information about the target applications but was later provided with the login credentials for different privilege levels for the Gray Box activity. This was done to simulate as closely as possible the viewpoint of a completely External as well as Internal attacker.

Test Cases were generated and Testing was done against industry best practices like Open Web Application Security Project (OWASP).

Some of the sample Test Cases generated in accordance with the OWASP v3 guidelines are given in Table 13.

Test Findings: A total of 4 vulnerabilities were found and exploited using the set of *Open Source/Free tools* given in Tables 9, 10, 11 and 12. The details of the vulnerabilities along with their *Severity, Impact* and *Recommendations* for *Remediation* are as follows:

1. **Name of Vulnerability :** Auto Complete Feature Turned ON
   Risk: Medium
   Abstract: Remember me functionality is seen as a convenience for the end users to prevent them the hassle of re-entering the username and the password each time they

Table 9 Open source/free tools—for automated static analysis

Name of tool	Operating system	Tool type	Language	Source
Flawfinder	Linux, Unix, Mac, Windows	Stand-alone script	C/C++	http://www.dwheeler.com/flawfinder
RATS	Linux, Unix, Windows	Stand-alone script	C/C++, Pearl, PHP, Python	https://www.fortify.com/ssa-elements/threat-intelligence/rats.html/
FindBugs	Linux, Unix, Windows	Stand-alone GUI application	JAVA	http://findbugs.sourceforge.net/
Pychecker	Linux, Unix, Windows	Stand-alone script	Python	http://pychecker.sourceforge.net
Pixy	Linux, Unix, Windows	Stand-alone script	PHP (XSS, SQLI)	http://pixybox.seclab.tuwien.ac.at/pixy
FxCop	Windows	Stand-alone application	.NET Framework CLR	http://www.microsoft.com/download/en/details.aspx?id=6544
OWASP SWAAT	Linux, Windows	Stand-alone GUI application	Xml (JSP,ASP, PHP)	https://www.owasp.org/index.php/Category:OWASP_SWAAT_Project/

Table 10 Pen source/free tools—for network penetration testing

Name of tool	Purpose	Operating system	Source
Nmap	Network Scanning	Linux, Unix, Mac OS X, Windows	http://www.nmap.org/
	Port Scanning		
	OS Detection		
Hping	Port Scanning	Linux, Unix, Mac OS X, Windows	http://www.hping.org/
	Remote OS Fingerprinting		
SuperScan	Detect open TCP/UDP Ports	Windows	http://www.mcafee.com/us/downloads/free-tools/superscan.aspx/
	Detect Services Running on Open Ports		
	Run WHOIS, PING and LOOKUP Queries.		
Xprobe2	Remote active OS Fingerprinting	Linux/Unix	http://www.net-security.org/software.php?id=231
	TCP Fingerprinting		
	Port Scanning		
P0f	OS Fingerprinting	Linux, Unix, Mac OS X, Windows	http://www.net-security.org/software.php?id=164
	Firewall Detection		
Httprint	Web Server Fingerprinting	Linux, Unix, Mac OS X, Windows	http://net-square.com/httprint/
	Detect Web enabled devices which do not have a server banner string.		
	SSL Detection.		
Nessus (Personal Edition)	Detect vulnerabilities that allow remote cracker to control or access sensitive data.	Linux, Unix, Mac OS X, Windows	http://www.tenable.com/products/nessus/
	Detect Misconfigurations, Default Passwords and the Denial of Services.		
Brutus	TELNET, FTP and HTTP password cracker.	Windows	http://download.cnet.com/Brutus/3000-2344_4-10455770.html/
Metasploit (Community Edition)	Develop and Execute Exploit Code against a remote target	Linux, Unix, Mac OS X, Windows	http://www.rapid7.com/products/metasploit/download.jsp
	Test the Vulnerabilities of Computer Systems		

use the application from a specific computer. The application does not prohibit this feature which makes the users browser memorize the user IDs and passwords.
OWASP Reference: OWASP-AT-006
Ease of Exploitation: Medium
Impact: This function is insecure by design and leaves the user exposed to attacks locally. Since the username and password stored using this functionality is not encrypted. Anyone with access to the user's browser can get hold of the credentials.
Recommendations: Set autocomplete="off" in the input tag.
For Example: <input type="text" name="signon" autocomplete="off" />
Alternatively Java scripts can be used to disable autocomplete.

1. **Name of Vulnerability :** Password policy not properly enforced
Risk: Low
Abstract: The application fails to enforce the password policy on user, as user is allowed to set only alphabetic passwords such as "password". On the other hand, the application does not maintain password history.
OWASP Reference: OWASP-AT-005
Ease of Exploitation: Hard
Impact: Because most users prefer passwords that they can easily remember, dictionary attacks are often an effective method for a malicious user to find a password in

Table 11 Open source/free tools - for web app penetration testing

Name of tool	Purpose	Operating system	Source
Nmap	Finding Web Servers	Linux, Unix, Mac OS X, Windows	http://www.nmap.org/
Fiddler	Web Debugging Proxy	Windows	http://www.fiddler2.com/fiddler2/
Nikto	Identify Web Server type, version and add ons.	Linux, Unix, Mac OS X, Windows	http://www.cirt.net/nikto2
	Detects common server misconfigurations, default, insecure, outdated server and programs.		
WebScarab	Interceptor	Linux, Unix, Mac OS X, Windows	https://www.owasp.org/index.php/Category:OWASP_WebScarab_Project
	Identifies new URLs on test target.		
	Session ID analyzer		
	Parameter Fuzzer		
W3af	Vulnerability Tester	Linux, Unix, Mac OS X, Windows	http://www.w3af.org/download/
	Interceptor		
	Fuzzer		
Arachni	Multiple Scans	Linux, Unis, Mac OS X	http://www.arachni-scanner.com/download/
	Self learning from HTTP responses		
	Performing Meta analysis		
	Identification of false positives.		
OWASP ZAP	Intercepting Proxy	Linux, Unix, Mac OS X, Windows	http://code.google.com/p/zaproxy/downloads/list
	Brute force scanner		
	Fuzzer		
	Dynamic SSL certificates		
Nessus (Personal Edition)	Vulnerability Detection	Linux, Unix, Mac OS X, Windows	http://www.tenable.com/products/nessus/
	Detection of Denial of Service.		
Vega	Automated crawler and Vulnerability Scanner	Linux, Unix, Mac OS X, Windows	http://www.subgraph.com/vega_download.php
	Intercepting Proxy		
	SSL MITM		
Skipfish	Performs reconnaissance	Linux, Unix, Mac OS X, Windows	http://code.google.com/p/skipfish/downloads/list
	Build Interactive site map		
Metasploit (Community Edition)	Develop and execute exploit code against target	Linux, Unix, Mac OS X, Windows	http://www.rapid7.com/products/metasploit/download.jsp/
	Test Vulnerabilities		

Table 12 Open source/free tools - for social engineering

Name of tool	Purpose	Operating system	Source
SET	Augment and simulate Social Engineering Attacks using payloads, email phishing e.t.c.	Linux, OpenBSD, Windows	https://www.trustedsec.com/downloads/social-engineer-toolkit/
MALTEGO	Graphical representation of relationships between people, groups, websites, Companies e.t.c Displays the confidential connections	Linux, Unix, Mac OS X, Windows	http://www.paterva.com/web6/products/download.php/

Table 13 Some sample test cases generated for VAPT in bank A

Category	Reference number	Test name	Vulnerability
Configuration Management Testing	OWASP-CM-001	SSL/TLS Testing	SSL Weakness
	OWASP-CM-002	DB Listener Testing	DB Listener Weak
	OWASP-CM-003	Infrastructure and Application configurationmanagement testing	Infrastructure and Application Configuration management weak
	OWASP-CM-004	Testing for File Extensions Handling	File Extensions Handling
	OWASP-CM-005	Old Backup and Unreferenced Files	Old Backup and Unreferenced Files
Business Logic Testing	OWASP-BL-001	Testing for Business Logic	By Passable Business Logic
Authentication Testing	OWASP-AT-001	Credentials Transport over an Encrypted Channel	Credentials Transport over an Encrypted Channel
	OWASP-AT-002	Testing for User Enumeration	User Enumeration
	OWASP-AT-003	Testing for Guessable (Dictionary) User Accounts	Guessable User Accounts
	OWASP-AT-004	Brute Force Testing	Credential Brute Forcing
	OWASP-AT-005	Testing for bypassing Authentication Schema	Bypassing Authentication Schema
Authorization Testing	OWASP-AZ-001	Testing for Path Traversal	Path Traversal
	OWASP-AZ-002	Testing for bypassing Authorization Schema	Bypassing Authorization Schema
	OWASP-AZ-003	Testing for Privilege Escalation	Privilege Escalation
Session Management	OWASP-SM-001	Testing for Session Management Schema	Bypassing Session Management Schema, Weak Session Tokens
	OWASP-SM-002	Testing for Cookies Attributes	Cookies are set not 'HTTP Only', 'Secure' and No Time Validity
	OWASP-SM-003	Testing for Session Fixation	Session Fixation
	OWASP-SM-004	Testing for Exposed Session Variables	Exposed Sensitive Session Variables
Data Validation Testing	OWASP-DV-001	Testing for Reflected Cross Site Scripting	Reflected XSS
	OWASP-DV-002	Testing for Stored Cross Site Scripting	Stored XSS
	OWASP-DV-003	Testing for DOM based Cross Site Scripting	DOM XSS
	OWASP-DV-004	Testing for Cross Site Flashing	Cross Site Flashing
	OWASP-DV-005	SQL Injection	SQL Injection
Denial of Service Testing	OWASP-DS-001	Testing for SQL Wildcards Attack	SQL Wildcard Vulnerability
	OWASP-DS-002	Locking Customer Accounts	Locking Customer Accounts
	OWASP-DS-003	Testing for DoS Buffer Overflows	Buffer Overflows
	OWASP-DS-004	User Specified Object Allocation	User Specified Object Allocation
	OWASP-DS-005	User Input as a Loop Counter	User Input as a Loop Counter
Web Services Testing	OWASP-WS-001	WS Information Gathering	N.A
	OWASP-WS-002	Testing WSDL	WSDL Weakness
	OWASP-WS-003	XML Structural Testing	Weak XML Structure
	OWASP-WS-004	XML Content Level Testing	XML Content
	OWASP-WS-005	HTTP GET Parameters/REST Testing	WS HTTP GET Parameters/REST
	OWASP-WS-006	Naughty SOAP attachments	WS Naughty SOAP Attachments
	OWASP-WS-007	Replay Testing	WS Replay Testing
AJAX Testing	OWASP-AJ-001	Testing AJAX	AJAX Weakness

significantly less time than they would with brute force attacks. As user can set password only with alphabets, such a weakly formed password may allow an attacker to break passwords in less time. This is violation to the password policy maintained.

Recommendations: Password policy should be properly enforced on all the users.

1. **Name of Vulnerability :** Internal IP Address Leak
   Risk: Low

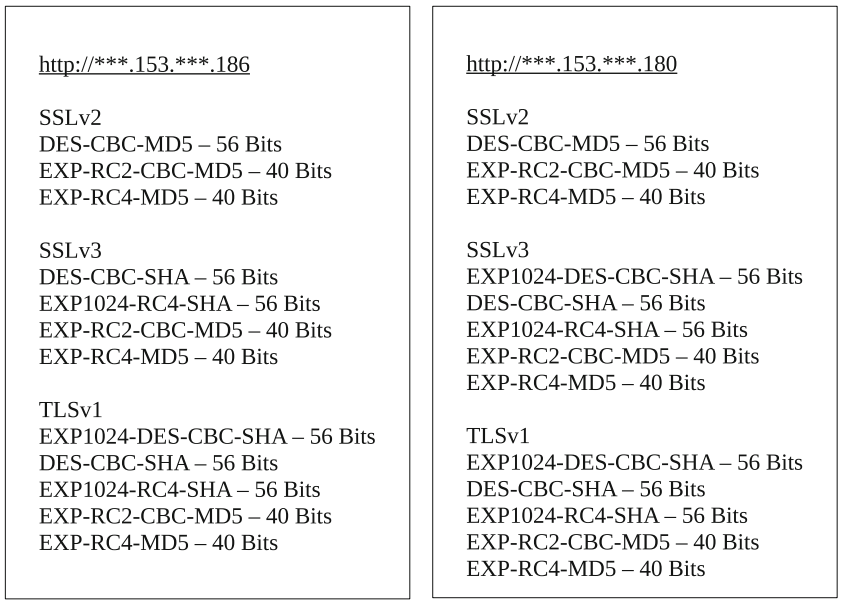

Fig. 15 Identified weak ciphers on web servers of target bank A

Abstract: Following Web Servers were found to reveal internal IP Addresses in response to the HTTP requests.

1. ***.153.***.186
2. ***.153.***.180

OWASP Reference: OWASP-CM-004
Ease of Exploitation: Hard
Impact: This should be avoided since it gives vital information to the attacker about the internal IP schema being followed by the Organization. This information can prove to be valuable in further attacks.
Recommendations: Please follow the link for Instructions on preventing this kind of Information Disclosure. http://support.microsoft.com/default.aspx?scid=KB%3BEN-US%3BQ218180&ID=KB%3BEN-US%3BQ218180

1. **Name of Vulnerability :** Weak SSL Ciphers found
Risk: Low
Abstract: Web Server was found Running SSL version 2 as well as weak ciphers (i.e, less than 128 bit) which are known to suffer from several cryptographic flaws.
Figure 15 shows the weak ciphers identified on web servers:
OWASP Reference: OWASP-CM-001
Ease of Exploitation: Hard
Impact: An Attacker may be able to exploit these issues to conduct man-in-middle attacks or deprecate communication between affected service and client.
Recommendations: Disable support for weak SSL Ciphers on web server if not required. This section, method, or task

5 Conclusion

In todays *Electronic Era*, *Anything* and *Everything* remains connected and exposed. The organizations need to develop an effective security infrastructure for *Security Assurance* against the increasing number of advanced exploits and hacking techniques.

VAPT is an efficient, cost effective and assured assessment tool to analyze the status of current security posture of an organization. It helps the organizations to remain protected from the outsider and insider threats. VAPT being *Proactive* in nature, enables an organization to know about the possible set of attacks even before their actual occurrence. Hence organization can take *Precautionary Steps* to safeguard its *Data resources* and *component systems* much before the attacker actually attacks.

In this paper we have described the four phases of vulnerability assessment as well as penetration testing. We have also described the phases and methodologies of VAPT . There are many commercial and Open Source tools available for VAPT, out of which we have shortlisted the best and the most popular list of *Open Source/Free Tools* for each category of testing in VAPT. We have also given the comparative analysis of all the techniques and methodologies used in VAPT along with the standards and precautions. A case study of a VAPT test conducted on a bank system using the shortlisted tools is also discussed.

References

1. Tiller, J.S.: CISO's Guide to Penetration Testing. CRC Press Publication, Boca Raton
2. The Canadian Institute of Chartered Accountants Information Technology Advisory Committee, Using an Ethical Hacking Technique to assess Information security Risk, Toronto. http://www.cica.ca/research-and-guidance/documents/it-advisory-committee/item12038.pdf. Accessed 03 Oct 2013
3. Xiong, P., Peyton, L.: A model driven penetration test framework for web applications. In: IEEE 8th Annual International Conference on Privacy, Security and Trust (2010)
4. Liu, B., Shi, L., Cai, Z.: Software vulnerability discovery techniques: a survey. In: IEEE 4th International Conference on Multimedia Information Networking and Security (2012)
5. Duan, B., Zhang, Y., Gu, D.: An easy to deploy penetration testing platform. In: IEEE 9th International Conference for young Computer Scientists (2008)
6. Dr. Geer, D., Harthorne, J.: Penetration testing: a duet. In: IEEE Proceedings of 18th Annual Computer Security Application Conference, ACSAC'02 (2002)
7. Sparks, S., Embleton, S., Cunningham, R., Zou, C.: Automated vulnerability analysis: leveraging control flow for evolutionary input crafting. In: IEEE 23rd Annual Computer Security Applications Conference (2007)
8. Open Web Application Security Project. OWASP Top 10 Project. http://www.owasp.org/index.php/category:OWASP_Top_Ten_Project. Accessed 03 Oct 2013
9. Turpe, S., Eichler, J.: Testing production systems safely: common precautions in penetration testing. In: IEEE Academy Industrial Conference (2009)
10. Halfold, W., Choudhary, S., Orso, A.: Penetration testing with improved input vector identification. In: IEEE International Conference on Software Testing Verification and Validation (2009)
11. Austin, A., Williams, L.: One technique is not enough: a comparison of vulnerability discovery techniques. In: IEEE International Symposium on Empirical Software Engineering and Measurement (2011)
12. The MITRE Corporation, Common Weakness Enumeration. http://www.cwe.mitre.org/. Accessed 03 Oct 2013
13. SANS Institute. SANS Top 25 Software Errors. http://www.sans.org/top25-software-errors/. Accessed 03 Oct 2013
14. Institute for Security and Open Methodologies. Open Source Security Testing Methodology Manual. http://www.isecom.org/mirror/OSSTMM.3.pdf. Accessed 03 Oct 2013
15. Payment Card Industry Security Standards. Payment Card Industry Data Security Standard. http://www.pcisecuritystandards.org/documents/pci_dss_v2.pdf. Accessed 03 Oct 2013
16. Open Web Application Security Project. OWASP Testing Guide. http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf. Accessed 03 Oct 2013
17. International Organization for Standardization. IEC/ISO 27001:2013. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54534. Accessed 03 Oct 2013
18. LanFang, W., HaiZhou, K.: A research of behavior based penetration testing model of the network. In: IEEE International Conference on Industrial Control and Electronics Engineering (2012)
19. iVolution Security Technologies. Benefits of Penetration Testing. http://www.ivolutionsecurity.com/pen_testing/benefits.php. Accessed 03 Oct 2013
20. Antunes, N., Vieira, M.: Benchmarking vulnerability detection tools for web services. In: IEEE International Conference on Web Services (2010)
21. White Hat Statistics Report' 2013. https://www.whitehatsec.com. Accessed 03 Oct 2013
22. Shah, S.: Vulnerability assessment and penetration testing (VAPT) techniques for cyber defence. IET-NCACNS' SGGS, Nanded (2013)
23. Shah, S., Mehtre, B.M.: A modern approach to cyber security analysis using vulnerability assessment and penetration testing. In: NCRTCST' 2013, Hyderabad, India
24. Shah, S., Mehtre, B.M.: School of Computer and Information Sciences, University of Hyderabad, Hyderabad, India. In: 2013 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)
25. McDermott, J.P.: Attack net penetration testing. In: Proceedings of the 2000 Workshop on New Security Paradigms. ACM Press, New York (2001)