

Web Application Security Whitepaper Response to Call for Papers from NASS Corporate Affiliates

Prepared for:

The National Association of Secretaries of State

Attn: Stacy Fisher

Assistant/Meeting Coordinator

Submitted by:

Eric Eifert

ManTech Security & Mission Assurance

1951 Kidwell Drive

Suite 500

Vienna, VA 22182

20 Jan 11

CONTENTS

1	What is Web Application Security?.....	3
2	Why Should the States Care?	4
3	Web Vulnerabilities	5
3.1	Input validation.....	5
3.2	Access Controls.....	5
3.3	Account and Session Management.....	6
3.4	Cross Site Scripting (XSS) Flaws.....	6
3.5	Buffer Overflows	7
3.6	Command Injection	7
3.7	Error Handling.....	8
3.8	Cryptography	8
3.9	Remote Administration.....	8
3.10	Misconfigurations	9
4	Why Look for Vulnerabilities?	10
5	Security Analysis Techniques	11
6	Conclusion	12
7	Works Consulted.....	13
8	Glossary	14

1 WHAT IS WEB APPLICATION SECURITY?

There are three major areas of Computer, or Cyber, Security.

- Network or Physical Level
- Operating System Level
- Application Level

For decades, the primary focus of most Cyber Security efforts was aimed at the first two levels; network level and operating system level. An enormous amount of time, effort, and money has been invested in discovering and correcting flaws in the computer networks and operating systems. Each subsequent iteration of these technologies brought new innovations, architectures, and methodologies to better protect the underlying systems and information. It is safe to say that network and operating system security, while not perfect, is a mature operating space. The same cannot be said for the third level of Cyber Security, the applications.

Until recently (roughly 5 years), applications were tightly integrated with the operating system. Such a model relied heavily on the underlying security of the operating system for protection of data. This model of individual applications installed on many clients and servers changed dramatically with the explosion of the Internet and Web Browsers. Applications no longer had to be installed on individual computers. At that point, anyone could access an application by using a standards-based web browser from anywhere in the world. On the server side, the web server was still tightly coupled to the operating system, but the higher order application logic is very much decoupled and abstracted from this foundation. Java, PHP, and other active code bases are operating system and web server agnostic. As long as the industry standards are followed, the web application code can run anywhere, on virtually any platform.

This decoupling has several important security implications. In the network and operating systems protection and security model, defensive measures primarily focus on protecting an “internal” trusted segment from the “outside” world, which is the rest of the Internet. This is accomplished through the use of a perimeter that consists of various appliances and network architectures which carefully filter traffic entering the protected interior. By their very nature, Web Applications cannot usually be protected in this manner. These applications generally need to be available to the entire Internet or a large subset of the public IP space. This implies allowing a direct pipe through all of the perimeter protection mechanisms directly to the application. Because of this paradigm shift, the traditional firewall, intrusion detection, and operating system protection are simply not effective. These applications require new methodologies and technologies to properly protect and secure the confidentiality, integrity, and availability of the data.

2 WHY SHOULD THE STATES CARE?

Since the explosion of Internet Web based applications in the past few years, more and more customers, such as residents and employees, expect and demand easy, simple access to information and services via the web. The ability to access information from anywhere, at anytime, by simply using a web browser is an extremely powerful idea. This model has even more recently expanded to mobile devices (cell phones, PDAs, laptops, netbooks, etc). Precisely because of the ubiquity and prevalence of these applications and the users that access them, there has been an explosion of vulnerabilities and associated exploits targeted against the gamut of web applications and underlying code. Because of the open nature of most applications and the sheer volume of users, it is easy for an attacker to hide malicious activities and remain undetected for long periods of time. Because many of these applications are custom developed, or at least modified, there are many new weaknesses introduced on an almost daily basis. Most are very trivial to exploit even without special tools or knowledge. The security industry has just now started to “catch up” and create standardization on security best practices and tools to counter these new threats.

The result is a “perfect storm” for States and the information they must collect, maintain, protect, and make available to their citizens. There is tremendous pressure, both legislative and public, to deliver more information and public services via the web. At the same time, many, if not most, of these applications are very vulnerable to attack and exploitation of the applications and underlying data being collected and processed. A good example of this “storm” is the Help America Vote Act (HAVA). The Federal Government mandated and provided funding for all states to develop and maintain voter registration records electronically after the 2000 election cycle. Every state designed, developed, and deployed custom solutions to meet this requirement. Most were web based applications that could be accessed by state officials and various state agencies to input and output the necessary data. Many of these systems were tested prior to deployment by ManTech and were found to have significant security weakness that allowed unauthorized access to, and modification of, the voter registration records. The flaws were corrected by the developers before the systems were used during actual election cycles. This vignette highlights two important points:

- Security flaws will exist in almost any new web application
- Proper vulnerability testing prior to launch provides a check and balance to ensure the software developers have corrected or mitigated any potential security issues

This example is simply one of many which states have already been, or soon will be, facing when bringing more services and information “online” to the public.

3 WEB VULNERABILITIES

There are ten broad categories of security flaws or vulnerabilities which are common to almost all web based applications. These vulnerabilities can be lumped into four major areas of concern:

- Authentication weaknesses
- Code weaknesses
- Cryptographic weaknesses
- Administration weaknesses

3.1 INPUT VALIDATION

Input Validation is perhaps the simplest and most common of all web application security flaws. Almost everyone who has ever used a web based application has at least inadvertently tested this vulnerability. Anytime information is entered into a web page, within a form, or in the URL itself, the application must accept that input and do something with it. Very often no checking is performed to verify the information at least notionally meets what is expected before the input is passed to the computer code for execution. A common example can be a date field. Legitimate date formats are very narrowly defined. An attacker, or a bad typist, can enter more or different data than would normally be expected for a date field. If the data is not validated prior to being passed to the application, several things may occur. The application may crash because too much data was entered. Malicious code could be embedded in the data entered, which can potentially be executed by the application or the underlying database. The application may simply process the information as entered and store incorrect data. All of these things may occur if the data is not explicitly checked prior to handing it off to the application for processing (OWASP, 2007).

Key Points:

- Check before you use anything in HTTPS request
- Canonicalize before you check
- Client-side validation can be circumvented
- Reject anything not specifically allowed
 - Type of data entry
 - Minimum and maximum lengths
 - Character type sets
 - Minimum and maximum values
 - Obfuscated format types

3.2 ACCESS CONTROLS

Access Controls is a broad category of security flaws that effect any web application that requires authentication prior to access to the data or the system. Proper authentication and session management is critical to web application security. Flaws in this area most frequently involve the failure to protect credentials and session tokens through their lifecycle. These flaws can lead to the hijacking of user or administrative accounts, undermine authorization and accountability controls, and cause privacy violations. The goal is to verify that the application properly authenticates users and properly protects identities and their associated credentials (OWASP, 2007).

Key Points:

- Thoroughly document your access control policy
- Don't use any "IDs" that an attacker can manipulate
- Implement access control in a centralized module

3.3 ACCOUNT AND SESSION MANAGEMENT

Account and session management includes all aspects of handling user accounts and managing active sessions. Even strong authentication mechanisms can be undermined by flawed account and credential management functions. These mechanisms include:

- Password changes
- Lost passwords
- Recalling passwords
- Account updates

User authentication on the web typically involves the use of a User ID and password. Stronger methods of authentication are commercially available such as software and hardware based cryptographic tokens or biometrics, but such mechanisms are more expensive to implement and maintain. A wide array of account and session management flaws can result in the compromise of user or system administration accounts. Web applications must establish sessions to keep track of the stream of requests from each user. The HTTP protocol does not provide this capability, so web applications must create it themselves. Frequently, the web application environment provides a session capability, but many developers prefer to create their own session tokens. In either case, if the session tokens are not properly protected, an attacker can hijack an active session and assume the identity of a user (OWASP, 2007).

Key Points:

- Keep credentials secret at all times
- Use only the random Session ID provided by your environment

3.4 CROSS SITE SCRIPTING (XSS) FLAWS

Cross-site scripting, better known as XSS, is a subset of HTML injection. XSS is one of the most prevalent web application security issues. XSS flaws occur whenever an application takes data that originated from a user and sends it to a web browser without first validating or encoding that content. This enables an attacker to redirect unsuspecting victims to a malicious web site in

order to steal credentials or other sensitive information. XSS is the underlying flaw that makes most email phishing attacks possible. XSS allows attackers to execute scripts in the victim's browser, which can hijack user sessions, deface web sites, insert hostile content, conduct phishing attacks, and take over the user's browser using scripting malware (OWASP, 2007).

Key Points:

- Never trust user input and always filter metacharacters

3.5 BUFFER OVERFLOWS

This category of flaws is related to the original operating system buffer overflows which are still fairly common. The same issues that plague operating systems can cause problems in web applications. Because web application must accept and process all manner of input from users, they are inherently vulnerable to attackers attempting to cause a memory buffer problem in the application. A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code (OWASP, 2007).

Key Points:

- If possible don't use the C or C++ programming languages
- Be careful about reading into buffers
- Use safe string libraries correctly

3.6 COMMAND INJECTION

This type of attack usually centers around the underlying web application database and its handling of SQL commands. Because of this, this type of flaw is commonly referred to as SQL injection. Command Injection flaws, particularly SQL injection, are common in web applications. There are many types of injections: SQL, LDAP, XPath, XSLT, HTML, XML, OS command injection, and so on. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. Attackers trick the interpreter into executing unintended commands via supplying specially crafted data. Injection flaws allow attackers to create, read, update, or delete any arbitrary data available to the application. In the worst case scenario, these flaws allow an attacker to completely compromise the application and the underlying systems (OWASP, 2007).

Key Points:

- Use extreme care when invoking an interpreter
- Use limited interfaces where possible, such as "PreparedStatement"
- Check return values

3.7 ERROR HANDLING

All web applications must handle error conditions that range from network issues, to flawed browser requests, to invalid user requests. When an error occurs, the application must convey the nature of the problem to the user. In doing so, applications often unintentionally reveal too much underlying information about their configuration and potential security flaws. Web applications will often leak information about their internal state through detailed or debug error messages. Often, this information can be leveraged to launch, or even to automate, more powerful attacks (OWASP, 2007).

Key Points:

- Make sure error screens don't print stack traces
- Design your error handling scheme carefully
- Configure your server properly

3.8 CRYPTOGRAPHY

This category actually encompasses two distinct flaws. Often, web applications do not encrypt sensitive traffic at all. This exposes the information to capture while in transit. Even if the traffic is encrypted, the underlying encryption mechanism or algorithm is either weak or improperly implemented, resulting in data leakage. In many cases, the encryption itself is fine, but the tokens or keys are inadvertently revealed, resulting in trivial decryption and exposure. Encryption also extends to data at rest, basically stored in files or a database. For particularly sensitive data, it should be encrypted on the server as well as on the network (OWASP, 2007).

Key Points:

- Use existing algorithm, no need to develop new ones
- Be extremely careful storing keys, certificates, and passwords
- Rethink whether you need to store the information
- Don't store user passwords – use a hash like SHA-512

3.9 REMOTE ADMINISTRATION

Because of the distributed nature of web applications, it is often necessary or desirable to have access to many different applications from one location or by one administrator. This implies the use of remote administrator tools and capabilities. Unfortunately, these mechanisms are most commonly implemented over the same public channels that users and attackers are using to access the application. This exposes those powerful administrative interfaces to attack and exploitation. If they are not configured properly, an attacker can gain direct administrative access to the site and the server, bypassing any application security measures and gaining access to all the data stored on the system. Stronger security mechanisms are warranted for application administration. Methods such as two factor authentication, out of band access, and biometrics should be considered for added security (OWASP, 2007).

Key Points:

- Eliminate all administration over the Internet
- Separate the admin application from the main application
- Limit the scope of remote administration

3.10 MISCONFIGURATIONS

All web applications and their underlying web servers have many security related configuration options and features. If not properly installed, configured, maintained, and audited a simple misconfiguration can easily lead to compromise. Configuration mistakes can include:

- Default accounts and passwords
- Uninstalled patches or upgrades
- Unnecessary default, backup, sample applications, and libraries
- Overly informative error messages
- Misconfigured SSL, default certificates, self-signed certificates
- Unused administrative services
- Lack of proper auditing

All of these problems are easily avoidable, but take a considerable effort in administration and maintenance to ensure they are not initially present or appear during production and use.

Key Points:

- Keep up with patches
- Use scanning tools, such as Nikto, Nessus, Web Scarab, and Web Inspect
- Harden all servers

4 WHY LOOK FOR VULNERABILITIES?

No one sets out to design a web application that is vulnerable to attack. Most web application developers are conscientious and concerned about security, but they have many other, often competing, requirements to consider. Developers have to make the application function according to business needs. They must make the application user friendly. They have to make the application visually pleasing and attractive. They must consider performance, data storage, redundancy, availability, etc. Security frequently gets shuffled to the back of the list during development.

To truly be secure, one must assume that flaws will make it into every web application. Even if not vulnerable on the day of release, it is almost inevitable that flaws will be discovered as time passes. To guard against this threat, an organization must periodically test for vulnerabilities and fix those that are discovered. This process should embrace a risk analysis approach to mitigation of discovered security flaws.

5 SECURITY ANALYSIS TECHNIQUES

There are four basic approaches to testing web applications for security flaws:

- Manual penetration testing
- Automated vulnerability scanning
- Manual code review
- Automated code analysis

The most common approach to finding vulnerabilities is to analyze the running application. The two techniques are “vulnerability scanning” (using tools and signature databases) and “penetration testing” (custom testing by experts). However, for many types of problems, analyzing the running application is very time-consuming and inaccurate. SQL injection, for example, is very difficult to find and diagnose in a running application, but can be quickly found by analyzing the source code.

The other approach is to analyze the source code. Like penetration testing, this can be done manually, a source code review, or with tools, static analysis. Code-based approaches have a reputation for being expensive and time-consuming, but this reputation is unfounded. For many types of issues, using the code is many times faster and more accurate than penetration testing.

The most cost-effective approach to application security is a “combined” or “integrated” approach. The assessor should be encouraged to use the most appropriate tool to find problems in the most cost-effective manner. For example, an assessor may notice a potential vulnerability during a penetration test, automatically scan the code for possible instances of the problem, and then confirm using code review.

Note that the purely automated approaches, the scanning and static analysis, are especially ineffective for application security. Most experts put the effectiveness of pure scanning or static analysis at less than 20%. This is largely due to the custom nature of applications. Because each one is different, there is no database of signatures the automated tools can use as a reference.

6 CONCLUSION

Web Applications are clearly the “new frontier” of cyber security. They are already prevalent and becoming more popular and pervasive every day. While Network and Operating System cyber security practitioners have had decades to perfect their techniques and methodologies, web applications are less than ten years old. This gives the attacker at a distinct advantage. We must carefully consider the most common and straightforward security issues first and work to perfect our approach to protecting applications against those attacks. With technology continuing to evolve at a rapid pace, and with more users, all potentially malicious, accessing the Internet and its applications every day, the threats and challenges to web application security will only continue to grow. Implementing a comprehensive web application security program will be critical to the success of any online presence in the next decade.

7 WORKS CONSULTED

OWASP. (2007, September). *Open Web Application Security Project Top 10 - A1 XSS*. Retrieved January 2009, from Open Web Application Security Project Top 10 Vulnerabilities: http://www.owasp.org/index.php/Top_10_2007-A1

OWASP. (2007, September). *Open Web Application Security Project Top 10 - A10 Failure to Restrict URL Access*. Retrieved January 2009, from Open Web Application Security Project Top 10 Vulnerabilities: http://www.owasp.org/index.php/Top_10_2007-A10

OWASP. (2007, September). *Open Web Application Security Project Top 10 - A2 Injection Flaws*. Retrieved January 2009, from Open Web Application Security Project Top 10 Vulnerabilities: http://www.owasp.org/index.php/Top_10_2007-A2

OWASP. (2007, September). *Open Web Application Security Project Top 10 - A3 Malicious File Execution*. Retrieved January 2009, from Open Web Application Security Project Top 10 Vulnerabilities: http://www.owasp.org/index.php/Top_10_2007-A3

OWASP. (2007, September). *Open Web Application Security Project Top 10 - A6 Information Leakage and Improper Error Handling*. Retrieved January 2009, from Open Web Application Security Project Top 10 Vulnerabilities: http://www.owasp.org/index.php/Top_10_2007-A6

OWASP. (2007, September). *Open Web Application Security Project Top 10 - A7 Broken Authentication and Session Management*. Retrieved January 2009, from Open Web Application Security Project Top 10 Vulnerabilities: http://www.owasp.org/index.php/Top_10_2007-A7

OWASP. (2007, September). *Open Web Application Security Project Top 10 - A9 Insecure Communications*. Retrieved January 2009, from Open Web Application Security Project Top 10 Vulnerabilities: http://www.owasp.org/index.php/Top_10_2007-A9

8 GLOSSARY

Canonicalize - In computer science, canonicalization (abbreviated c14n, where 14 represents the number of letters between the C and the N) is a process for converting data that has more than one possible representation into a "standard" canonical representation. This can be done to compare different representations for equivalence, to count the number of distinct data structures, to improve the efficiency of various algorithms by eliminating repeated calculations, or to make it possible to impose a meaningful sorting order.

(source: Wikipedia.org)

Cookie - HTTP cookies, more commonly referred to as [Web](#) cookies, tracking cookies or just cookies, are parcels of text sent by a [server](#) to a Web [client](#) (usually a [browser](#)) and then sent back unchanged by the client each time it accesses that server. [HTTP](#) cookies are used for [authenticating](#), session tracking (state maintenance), and maintaining specific information about users, such as site preferences or the contents of their [electronic shopping carts](#). The term "cookie" is derived from "[magic cookie](#)," a well-known concept in [UNIX](#) computing which inspired both the idea and the name of HTTP cookies.

Because they can be used for tracking browsing behavior, cookies have been of concern for [Internet privacy](#). As a result, they have been subject to legislation in various countries such as the [United States](#), as well as the [European Union](#). Cookies have also been criticized because the identification of users they provide is not always accurate and because they could potentially be a target of network attackers. Some alternatives to cookies exist, but each has its own uses, advantages, and drawbacks.

Cookies are also subject to a number of misconceptions, mostly based on the erroneous notion that they are [computer programs](#). In fact, cookies are simple pieces of data unable to perform any operation by themselves. In particular, they are neither [spyware](#) nor [viruses](#), although cookies from certain sites are described as spyware by many anti-spyware products because they allow users to be tracked when they visit various sites.

Most modern browsers allow users to decide whether to accept cookies, but rejection makes some [websites](#) unusable. For example, shopping carts implemented using cookies do not work if cookies are rejected.

(source: Wikipedia.org)

Cross site scripting (XSS) - Cross-site scripting (XSS) is a type of [computer security vulnerability](#) typically found in [web applications](#) which allow [code injection](#) by malicious web users into the [web pages](#) viewed by other users. Examples of such code include [HTML](#) code and [client-side scripts](#). An exploited cross-site scripting vulnerability can be used by attackers to bypass [access controls](#) such as the [same origin policy](#). Vulnerabilities of this kind have been

exploited to craft powerful [phishing](#) attacks and browser exploits. As of 2007, cross-site scripting carried out on websites were roughly 80% of all documented security vulnerabilities.^[1] Often during an attack "everything looks fine" to the [end-user](#)^[2] who may be subject to unauthorized access, theft of sensitive data, and financial loss.^[3]

(source: Wikipedia.org)

IP address - An [Internet Protocol](#) (IP) address is a numerical identification ([logical address](#)) that is assigned to devices participating in a [computer network](#) utilizing the [Internet Protocol](#) for communication between its nodes.^[1] Although IP addresses are stored as [binary numbers](#), they are usually displayed in [human-readable](#) notations, such as 208.77.188.166 (for [IPv4](#)), and 2001:db8:0:1234:0:567:1:1 (for [IPv6](#)). The role of the IP address has been characterized as follows: "A [name](#) indicates what we seek. An address indicates where it is. A route indicates how to get there." ^[2]

The original designers of TCP/IP defined an IP address as a [32-bit](#) number^[1] and this system, now named [Internet Protocol Version 4](#) (IPv4), is still in use today. However, due to the enormous growth of the Internet and the resulting depletion of the address space, a new addressing system ([IPv6](#)), using 128 bits for the address, was developed ([RFC 1883](#)).

The Internet Protocol also has the task of [routing](#) data [packets](#) between networks, and IP addresses specify the locations of the source and destination nodes in the [topology](#) of the [routing](#) system. For this purpose, some of the bits in an IP address are used to designate a [subnet](#). (In [CIDR notation](#), the number of bits used for the subnet follows the IP address. E.g. 208.77.188.166/24) An IP address can be private, for use on a [LAN](#), or public, for use on the Internet or other [WAN](#).

Early specifications intended IP addresses to each be uniquely assigned to a particular computer or device.^[citation needed] However, it was found that this was not always necessary as [private networks](#) developed and address space needed to be conserved ([IPv4 address exhaustion](#)). [RFC 1918](#) specifies private address spaces (also known as non-routable addresses) that may be reused by anyone; today, such private networks typically connect to the Internet through [Network Address Translation](#) (NAT). In addition, technologies such as [anycast](#) addressing have been developed to allow multiple hosts at the same IP address but in different portions of the Internet to service requests by network clients.

The [Internet Assigned Numbers Authority](#) (IANA) manages the global IP address space. IANA works in cooperation with five [Regional Internet Registries](#) (RIRs) to allocate IP address blocks to [Local Internet Registries](#) (Internet service providers) and other entities.

(source: Wikipedia.org)

Malware - Malware, a [portmanteau](#) from the words [malicious](#) and [software](#), is software designed to infiltrate or damage a computer system without the owner's [informed consent](#). The expression is a general term used by computer professionals to mean a variety of forms of hostile, intrusive, or annoying software or program code.^[1] The term "[computer virus](#)" is sometimes used as a catch-all phrase to include all types of malware, including true viruses.

Software is considered malware based on the perceived intent of the creator rather than any particular features. Malware includes [computer viruses](#), [worms](#), [trojan horses](#), most [rootkits](#), [spyware](#), dishonest [adware](#), [crimeware](#) and other malicious and unwanted software. In [law](#), malware is sometimes known as a computer contaminant, for instance in the legal codes of several American states, including [California](#) and [West Virginia](#).^[2] ^[3]

Malware is not the same as defective software, that is, software which has a legitimate purpose but contains harmful [bugs](#).

Preliminary results from [Symantec](#) published in 2008 suggested that "the release rate of malicious code and other unwanted programs may be exceeding that of legitimate software applications."^[4] According to [F-Secure](#), "As much malware [was] produced in 2007 as in the previous 20 years altogether."^[5] Malware's most common pathway from criminals to users is through the [Internet](#), by email and the [World Wide Web](#).^[6]

(source: Wikipedia.org)

One Time Password (OTP) - The purpose of a one-time password (OTP) is to make it more difficult to gain unauthorized access to restricted resources, like a computer account. Traditionally [static passwords](#) can more easily be accessed by an unauthorized intruder given enough attempts and time. By constantly altering the password, as is done with a one-time password, this risk can be greatly reduced.

There are basically three types of one-time passwords: the first type uses a mathematical algorithm to generate a new password based on the previous, a second type that is based on time-synchronization between the authentication server and the client providing the password, and a third type that is again using a mathematical algorithm, but the new password is based on a challenge (e.g. a random number chosen by the authentication server or transaction details) and a counter instead of being based on the previous password.

(source: Wikipedia.org)

Phishing - In the field of computer security, phishing is the [criminally fraudulent](#) process of attempting to acquire sensitive information such as usernames, [passwords](#) and credit card details by masquerading as a trustworthy entity in an electronic communication. Communications purporting to be from popular social web sites ([YouTube](#), [Facebook](#), [MySpace](#), [Windows Live Messenger](#)), auction sites ([eBay](#)), [online banks](#) ([Wells Fargo](#), [Bank of America](#), [Chase](#)), online payment processors ([PayPal](#)), or IT Administrators ([Yahoo](#), [ISPs](#), corporate) are commonly used to lure the unsuspecting. Phishing is typically carried out by [e-mail](#) or [instant messaging](#).^[1] and it often directs users to enter details at a fake website whose [look and feel](#) are almost identical to the legitimate one. Even when using [server authentication](#) it requires skill to detect that the website is fake. Phishing is an example of [social engineering](#) techniques used to fool users ^[2], and exploits the poor usability of current web security technologies ^[3]. Attempts to deal with the growing number of reported phishing incidents include [legislation](#), user training, public awareness, and technical security measures.

A phishing technique was described in detail in 1987, and the first recorded use of the term "phishing" was made in 1996. The term is a variant of fishing,[\[4\]](#) probably influenced by [phreaking](#),[\[5\]](#)[\[6\]](#) and alludes to baits used to "catch" financial information and passwords.

(source: Wikipedia.org)

Session - In [computer science](#), in particular [networking](#), a session is a semi-permanent interactive information exchange, also known as a dialogue, a conversation or a meeting, between two or more communicating devices, or between a computer and user (see [Login session](#)). A session is set up or established at a certain point in time, and torn down at a later point in time. An established communication session may involve more than one message in each direction. A session is typically, but not always, [stateful](#), meaning that at least one of the communicating parts need to save information about the session history in order to be able to communicate, as opposed to [stateless](#) communication, where the communication consists of independent requests with responses.

Communication sessions may be implemented as part of protocols and services at the [application layer](#), at the [session layer](#) or at the [transport layer](#) in the [OSI model](#).

Application layer examples:

[HTTP](#) sessions, which may allow [dynamic web pages](#), i.e. interactive web pages, as opposed to [static web pages](#).

A [telnet](#) remote login session

[Session layer](#) example:

A [Session Initiation Protocol](#) (SIP) based [Internet phone](#) call

Transport layer example:

A [TCP](#) session, which is synonymous to a TCP [virtual circuit](#), a TCP [connection](#), or an established TCP [socket](#).

In the case of transport protocols which do not implement a formal session layer (e.g., [UDP](#)) or where sessions at the session layer are generally very short-lived (e.g., [HTTP](#)), sessions are maintained by a higher level program using a method defined in the data being exchanged. For example, an HTTP exchange between a browser and a remote host may include an [HTTP cookie](#) which identifies state, such as a unique [session ID](#), information about the user's preferences or authorization level.

Protocol version [HTTP/1.1](#) makes it possible to reuse the same TCP session for a sequence of service requests and responses (a sequence of file transfers) in view to reduce the session establishment time, while [HTTP/1.0](#) only allows a single request and response during one TCP session. However, this transport layer session mechanism should not be confused with a so called HTTP session, since it is not lasting sufficiently long time, and does not provide application level interactive services such as dynamic web pages.

(source: Wikipedia.org)

Session Hijack - The term session hijacking refers to the exploitation of a valid [computer session](#) - sometimes also called a session key - to gain unauthorized access to information or services in a computer system. In particular, it is used to refer to the theft of a [magic cookie](#) used to authenticate a user to a remote server. It has particular relevance to [web developers](#), as the [HTTP cookies](#) used to maintain a session on many web sites can be easily stolen by an attacker using an intermediary computer or with access to the saved cookies on the victim's computer (see [HTTP cookie theft](#)).

(source: Wikipedia.org)