Proposed New Vulnerabilities

7.Z Download of Code Without Integrity Check [DLB]

7.Z.1 Description of application vulnerability

Some applications download source code or executables from a remote location and use the source code or invokes the executables without sufficiently verifying the origin and integrity of the downloaded files.

7.Z.2 Cross reference

CWE:

494. Download of Code Without Integrity Check

7.Z.3 Mechanism of failure

An attacker can execute malicious code by compromising the host server, performing DNS spoofing, or modifying the code in transit.

7.Z.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Perform proper forward and reverse DNS lookups to detect DNS spoofing. Encrypt the code with a reliable encryption scheme before transmitting.
- This is only a partial solution since it will not prevent your code from being modified on the hosting site or in transit.
- Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
- Specifically, it may be helpful to use tools or frameworks to perform integrity checking on the transmitted code.
- If providing code that is to be downloaded, such as for automatic updates of software, then use cryptographic signatures for the code and modify the download clients to verify the signatures.

7.Y Incorrect Authorization [BJE]

7.Y.1 Description of application vulnerability

The software performs an inefficient or flawed authorization check when an actor attempts to access a resource or perform an action. This allows attackers to bypass intended access restrictions.

7.Y.2 Cross reference

CWE:

863. Incorrect Authorization

7.Y.3 Mechanism of failure

Authorization is the process of determining whether that user can access a given resource, based on the user's privileges and any permissions or other access-control specifications that apply to the resource.

When access control checks are incorrectly applied, users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information exposures, denial of service, and arbitrary code execution.

7.Y.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Ensure that you perform access control checks related to your business needs. These checks may be different and more detailed than those applied to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

7.X Inclusion of Functionality from Untrusted Control Sphere [DHU]

7.X.1 Description of application vulnerability

The software imports, requires, or includes executable functionality (such as a library) from a source that is outside of the intended set of resources and behaviour that are accessible to a single actor.

7.X.2 Cross reference

CWE:

829. Inclusion of Functionality from Untrusted Control Sphere

7.X.3 Mechanism of failure

When including third-party functionality, such as a web widget, library, or other source of functionality, the software must effectively trust that functionality. Without sufficient protection mechanisms, the functionality could be malicious in nature (either by coming from an untrusted source, being spoofed, or being modified in transit from a trusted source). The functionality might also contain its own weaknesses, or grant access to additional functionality and state information that should be kept private to the base system, such as system state information, sensitive application data, or the DOM of a web application.

This might lead to many different consequences depending on the included functionality, but some examples include injection of malware, information exposure by granting excessive privileges or permissions to the untrusted functionality, DOM-based XSS vulnerabilities, stealing user's cookies, or open redirect to malware.

7.X.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
- When the set of acceptable objects, such as filenames or URLs, is limited or known, create a mapping from a set of fixed input values (such as numeric IDs) to the actual filenames or URLs, and reject all other inputs.
- For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt". Features such as the ESAPI AccessReferenceMap provide this capability.
- For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.

7.W Improper Restriction of Excessive Authentication Attempts [WPL]

7.W.1 Description of application vulnerability

The software does not implement sufficient measures to prevent multiple failed authentication attempts within in a short time frame, making it more susceptible to brute force attacks.

7.W.2 Cross reference

CWE:

307. Improper Restriction of Excessive Authentication Attempts

7.W.3 Mechanism of failure

In a recent incident an attacker targeted a member of a popular social networking sites support team and was able to successfully guess the member's password using a brute force attack by guessing a large number of common words. Once the attacker gained access as the member of the support staff, he used the administrator panel to gain access to a number of accounts that belonged to celebrities and politicians. Ultimately, fake messages were sent that appeared to come from the compromised accounts.

7.W.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following way:

- Disconnecting the user after a small number of failed attempts

- Implementing a timeout
- Locking out a targeted account
- Requiring a computational task on the user's part.
- Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
- Consider using libraries with authentication capabilities such as OpenSSL or the ESAPIAuthenticator.

7.V URL Redirection to Untrusted Site ('Open Redirect') [PYQ]

7.V.1 Description of application vulnerability

A web application accepts a user-controlled input that specifies a link to an external site, and uses that link in a redirect without checking that the URL points to a trusted location. This simplifies phishing attacks.

7.V.2 Cross reference

CWE:

601. URL Redirection to Untrusted Site ('Open Redirect')

7.V.3 Mechanism of failure

An http parameter may contain a URL value and could cause the web application to redirect the request to the specified URL. By modifying the URL value to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials. Because the server name in the modified link is identical to the original site, phishing attempts have a more trustworthy appearance.

7.V.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Input Validation
- Assume all input is malicious. Use an "accept known good" input validation strategy, for example, use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (for example, do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
- When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if you are expecting colors such as "red" or "blue." Use a whitelist of approved URLs or domains to be used for redirection.

7.U Uncontrolled Format String [LHS]

7.U.1 Description of application vulnerability

The software uses externally-controlled format strings in `printf()` style functions, which can lead to buffer overflows or data representation problems.

7.U.2 Cross reference

CWE:

134. Uncontrolled Format String

7.U.3 Mechanism of failure

The programmer rarely intends for a format string to be user-controlled at all. This weakness frequently occurs in code that constructs log messages, where a constant format string is omitted.

In cases such as localization and internationalization, the language-specific message repositories could be an avenue for exploitation, but the format string issue would be resultant, since attacker control of those repositories would also allow modification of message length, format, and content.

7.U.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Ensure that all format string functions are passed as static string which cannot be controlled by the user and that the proper number of arguments is always sent to that function as well. If at all possible, use functions that do not support the `%n` operator in format strings.

7.T Use of a One-Way Hash without a Salt [MVX]

7.T.1 Description of application vulnerability

The software uses a one-way cryptographic hash against an input that should not be reversible, such as a password, but the software does not also use a salt[1] as part of the input.

7.T.2 Cross reference

CWE:

327. Use of a Broken or Risky Cryptographic Algorithm
759. Use of a One-Way Hash without a Salt

7.T.3 Mechanism of failure

This makes it easier for attackers to pre-compute the hash value using dictionary attack techniques such as rainbow tables.

7.T.4 Avoiding the vulnerability or mitigating its effects

Software developers can avoid the vulnerability or mitigate its ill effects in the following ways:

- Generate a random salt each time you process a new password. Add the salt to the plaintext password before hashing it. When you store the hash, also store the salt. Do not use the same salt for every password that you process.
- Use one-way hashing techniques that allow you to configure a large number of rounds, such as bcrypt. This may increase the expense when processing incoming authentication requests, but if the hashed passwords are ever stolen, it significantly increases the effort for conducting a brute force attack, including rainbow tables. With the ability to configure the number of rounds, one can increase the number of rounds whenever CPU speeds or attack techniques become more efficient.
- When industry-approved techniques are used, they must be used correctly. Never skip resource-intensive steps (CWE-325). These steps are often essential for preventing common attacks.

[1] In cryptography, a salt consists of random bits, early systems used a 12-bit salt, modern implementations use 48 to 12 bits.