Introduction to the OWASP Top 10 – 2021

Risk A8: Software and Integrity Failures

Key Concepts

Key Concept 1: Software Integrity Verification

- Process of verifying the integrity of software updates, third party libraries and other third-party software sources
- There are a variety of methods that can be used to verify the integrity of third-party software such as digital signatures, and download hashes

Key Concept 2: Continuous Integration

- Developers commit code is small segments, often doing so many times a day
- Automated “pipelines” build and test code submissions before they are fully merged into the project
- Be careful, you could pull in untrusted software libraries into your software during this process

Key Concept 3: Continuous Delivery

- Process of releasing software in an automated way, without manual intervention
- Be careful, you could pull in untrusted software libraries into your software during this process

Additional Information

Software Integrity Verification is based on the principle of trust but verify. This is the process of verifying the inclusion of functionality from untrusted software sources. Your application relies on libraries and frameworks that have not been written by you. The process of verifying the source of these libraries is part of Software Integrity Verification.

Continuous Integration refers to automated tasks of building and testing developer code submissions before merging them into the main project. If you are not careful, during Continuous Integration, you might also be pulling in untrusted software libraries.

Continuous Delivery is the process of releasing software in an automated way, without manual intervention. Again, if you are not careful during Continuous Delivery, you might be pulling in untrusted software libraries into your software.

Definition of Software and Integrity Failure

Having software code that does not prevent the inclusion of functionality from untrusted sources.

- Know, for your software application, what sources of code you trust
  - Think of the repositories of the libraries you are using and make sure they are mainstream
  - Always download updated source code dependencies from trusted and mainstream software repositories
- Be careful when running third party code *of any kind*
  - Know what software code is being pulled in dynamically and make sure to verify the integrity before use
  - Also, try to avoid deserializing untrusted data, a way for attackers to run untrusted code

Additional Information

The definition of A8 of 2021 – Software and Data Integrity Failures – is having software code that does not prevent the inclusion of functionality from untrusted sources without integrity verification.

This means that you need to know, for your software application what sources of code you trust. Think of repositories of libraries you are using, always download or update source code dependencies from software repositories and perform an integrity check on third party code before using it.

There are also cases of parsing data relating to software being loaded during runtime. An example of this is deserializing untrusted data at runtime within your web application. You need to know what software code is being pulled in dynamically by your web application or API. Dynamic loading of software can also cause Software and Data Integrity Failures.

Example

To demonstrate our good and bad example, consider our web application having a build-in update functionality. Within this update functionality we have hardcoded an HTTPS URL to download the latest software update.

Bad Example

```
Update() {
  var data = download("https://manicode.com/update.sh");
  exec(data);
}
```

Good Example

```
Update() {
  var data = download("https://manicode.com/update.sh");
  verifyHash(data, "e7de35ebe643d7a$d23fd814639ac420fc2a9b6");
  verifyGitContent(data);
  exec(data);
}
```

Additional Information

It turns out that putting your entire trust on the HTTPS URL may not be enough. This is because as one example, the operating system which is making the HTTPS could have been manipulated. Manipulation could have been done at a certificate, DNS, or another level outside of the control of our software update. Thus, the code you see in red above is not enough to protect us from a Software Download Integrity Failure.

A good example is to download the update via HTTPS, and then check the download to have a secure checksum value that you expect. This method, a hash commit check, is often used as an additional integrity check on the download so to prevent a Software Download Integrity Failure.

Challenges

- Many software solutions/components auto-update without sufficient integrity checks
  - Attackers know this and try to exploit it
  - Attackers target software update mechanisms
  - As a trend, problems around auto-updates are increasing

- Software integrity problems are hard to detect
  - Your developers are downloading software libraries required for the software they are making
  - Security teams are not involved with those processes, but they should be

Additional Information

We now understand how the risk of software and data integrity failures can materialize, let us look at why it is a common issue.

Many software solutions or software components auto-update without sufficient integrity checks. Attackers know this and try to exploit it, by targeting software update mechanisms. As a trend, in recent years, problems around auto-updates are increasing.

This risk is common also because software integrity problems are hard to detect. Developers are downloading software libraries required for the software they are making. Security teams are not involved with those processes enough, but they should be.

As we saw when we learned about A6 – Vulnerable and Outdated Components, keeping your third-party components up to date is important so not be exposed to the latest vulnerabilities. Adding to that, we need to perform software and data integrity checking for what software we do trust.

Best Protection Strategies

LEVAN

Learn enough cryptography to verify integrity of downloads
Ensure all third-party software and frameworks are updated
Verify software updates independently, using cryptography
Apply and use digital signatures
Note what software sources you trust

Additional Information

Above you have some of the best protection strategies against A8 of the Top 10 – Software and Data Integrity Failures