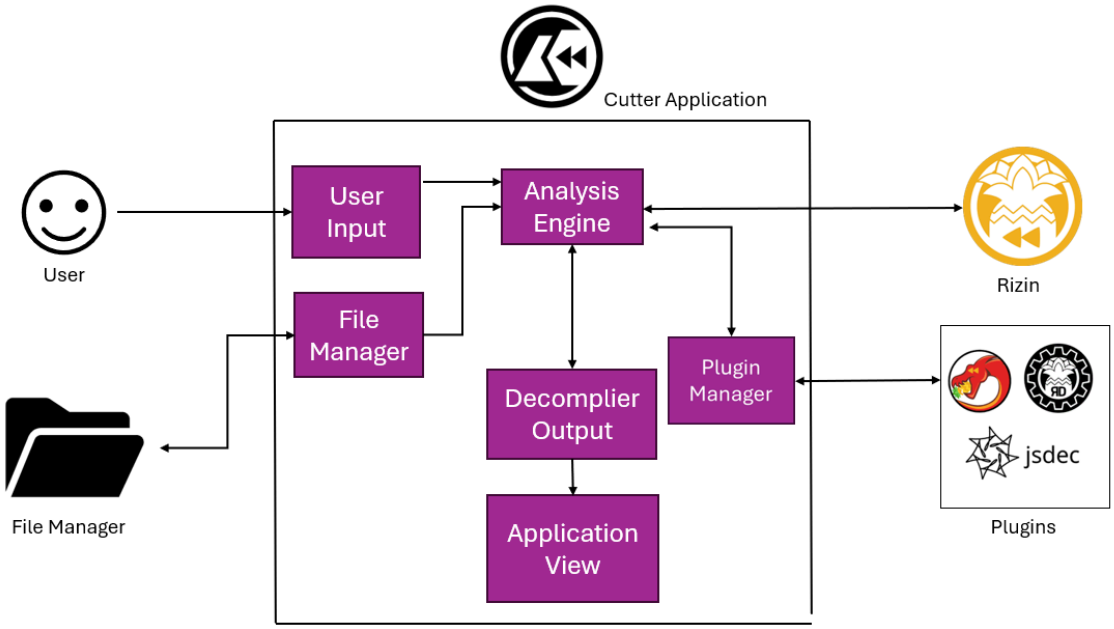
Deliverable: Security Design Review

Data Diagram

Cutter itself is not an online application, that is after downloading the application, it is self contained, communicating only with libraries and the User. It uses external dependencies such as Rizin and various plugins, but there is no true threat coming from the server. The primary paths are between Rizin/Plugins and Cutter, and the User and Cutter.

The primary paths are the User input to Cutter, where it is then handled by the cutter applications, and Cutter communicating with Rizin and its plugins to provide the decompiled view to the user of their binary file.



Threats to Users and Developers

- Threat Model

Being a tool to combat Malware attacks and improve the security of code by finding Malware and using the principles of Reverse Engineering to stop Malware, attackers have a vested interest in ensuring Cutter's either demise or corruption. Not only does the demise of Cutter make their jobs easier, but it ensures that their methods can stay hidden for longer.

- Threat 1. Compromised Legitimate Packages

Cutter, being an open source application, relies on a lot of other packages, dependencies, and even other open source projects to function. Some examples would be the Ghidra Decomplier and Rizin itself. If the base package becomes compromised from an attack, that attack can ripple its way down to Cutter, causing a security issue within Cutter itself. While this is generally a mute issue for open source software, it remains one of the biggest threats to cutter from a security standpoint.

- Threat 2. Code Injections from Malicious Sources

The previous threat talks about an attack rippling down the chain to Cutter. This issue speaks more about if contributed code is compromised in some way. Whether it be sneaky malware, or a virus that attaches itself to the various documents Cutter contains, there are plenty of available avenues for Cutter to be attacked from. Again, while uncommon, this is one of the biggest threats to Cutter, mainly to ensure that their application doesn't get corrupted, and then can become a source for the other dependencies that Cutter uses.

Mitigation for Threats

- Threat 1 Mitigation: Alternatives and Community

When a package or dependency becomes compromised, Cutter has two options: either to use a different version of the package/dependency or quickly switch away from that package/dependency if viable. Cutter must be in constant communication with the packages and dependencies it uses so it can ensure that any malware does not leak its way down into Cutter, which is where the community comes in. The community is an excellent source to find these security vulnerabilities, whether they are potential or actual.

- Threat 2 Mitigation: Version Control & Community

For the same reason that this threat exists, its solution also exists. If there is an attack on Cutter, then there are always previous versions of the app to fall back on. Whether it be on one's personal machine or github. However, damage could already be done for the hundreds that may have downloaded the application in its compromised state, which is what makes the open source community the next most powerful tool of mitigation. The community can be very vocal about vulnerabilities in code, both potential and real, so it makes solving malware attacks easier than a stand alone project. Also, Cutter has an intense reviewing process, and being an application to find security vulnerabilities, it is more than equipped to find vulnerabilities within its own code.

- Mitigations Conclusion

If there is a threat to the security of Cutter, Cutter has a Security policy to directly let the developers of the application know something has gone haywire and there needs to be a fix. All reported security issues can be emailed to security@cutter.re. While their security team is small, it is more than capable of ensuring that Cutter stays safe and is able to provide its services to its users.