Polyverse Boost Source Analysis Details: ./api/gin/gomerrrenderhandler.go

Date Generated: Thursday, September 7, 2023 at 3:15:17 AM PDT

Boost Architectural Quick Summary Security Report

Last Updated: Friday, September 8, 2023 at 3:38:37 PM PDT

Executive Level Report:

1. **Architectural Impact**: The project is a Go library that focuses on constraint handling and validation. The architecture is sound and follows Go's idiomatic style and structure for a library. However, there is a significant issue in the `gomerrrenderhandler.go` file where the function panics when it encounters an error during rendering. This could potentially cause the entire application to crash, leading to a Denial of Service (DoS) attack. This issue needs to be addressed to ensure the stability and reliability of the library.

2. **Risk Analysis**: The risk associated with the project is moderate. The improper error handling in the `gomerrrenderhandler.go` file is a significant risk that could lead to application crashes and potential information disclosure. This issue affects 100% of the project files reviewed. However, it's important to note that this analysis is based on a single file, and the actual risk may vary depending on the size and complexity of the entire project.

3. **Potential Customer Impact**: The improper error handling could lead to application instability and potential data leaks, which could negatively impact the customer's trust and usage of the library. It's recommended to handle the error gracefully, log the error, and return a generic error message to the client to mitigate this impact.

4. **Overall Issues**: The main issue identified in the project is improper error handling. This issue is categorized as a warning, indicating that it's a significant problem that needs to be addressed. The issue affects the `gomerrrenderhandler.go` file, which is 100% of the project files reviewed.

5. **Risk Assessment**: Based on the analysis of the single file provided, 100% of the project files have issues. This indicates a high risk to the overall health of the project source.

However, it's important to note that this risk assessment may not be representative of the entire project as it's based on a single file.

In conclusion, while the project follows Go's idiomatic style and structure for a library, the improper error handling issue identified poses a significant risk to the stability and reliability of the library. It's recommended to address this issue to ensure the overall health and success of the project.

Boost Architectural Quick Summary Performance Report

Last Updated: Thursday, September 7, 2023 at 3:14:44 AM PDT

Executive Report

Architectural Impact and Risk Analysis

1. **Reflection Usage in Go**: The file `api/gin/gomerrrenderhandler.go` has been flagged for using reflection in Go, which is computationally expensive and can slow down the execution of the program. This could potentially impact the performance of the software, especially under high load conditions. This is a medium risk issue as it affects the CPU usage of the software.

2. **Memory Usage**: The same file `api/gin/gomerrrenderhandler.go` also has a warning related to memory usage. This could potentially lead to memory leaks or inefficient memory usage, which could impact the overall performance and stability of the software. This is a high risk issue as it directly affects the stability of the software.

Potential Customer Impact

3. **Performance Degradation**: The issues identified could potentially lead to performance degradation, which could impact the user experience. This could result in slower response times and decreased throughput.

4. **Stability Issues**: The memory usage issue could potentially lead to stability issues, such as crashes or unexpected behavior. This could impact the reliability of the software and could lead to customer dissatisfaction.

Overall Health of the Project Source

5. **Percentage of Files with Issues**: Based on the provided information, only one file out of the entire project has been flagged with issues. This suggests that the majority of the project is well-structured and follows good practices. However, the issues identified in this one file are of high severity and should be addressed promptly.

Risk Assessment

6. **Risk Level**: Given that only one file has been flagged with issues, the overall risk level of the project is relatively low. However, the severity of the issues identified in this file is high, and if left unaddressed, could potentially impact the overall performance and stability of the software.

In conclusion, while the overall health of the project appears to be good, the issues identified in the `api/gin/gomerrrenderhandler.go` file should be addressed promptly to prevent potential performance degradation and stability issues.

Boost Architectural Quick Summary Compliance Report

Last Updated: Thursday, September 7, 2023 at 3:16:09 AM PDT

Executive Report: Software Project Analysis

Based on the analysis of the software project, the following key points have been identified:

1. **Architectural Impact**: The project appears to be a server-side application, likely a web API, with a focus on data handling, constraints, and error management. It is written in Go and follows a modular and structured approach. The project structure seems consistent with Go server applications, and no specific architectural problems were identified.

2. **Risk Analysis**: The most severe issue identified is related to GDPR/HIPAA compliance in the file `api/gin/gomerrrenderhandler.go`. The error handling mechanism logs and propagates all errors without filtering or anonymizing potentially sensitive data. This could lead to exposure of sensitive data like Personal Identifiable Information (PII) or Protected Health Information (PHI) which is a violation of GDPR and HIPAA. This represents a significant risk that needs to be addressed immediately.

3. **Potential Customer Impact**: If not addressed, the GDPR/HIPAA compliance issue could lead to data breaches, resulting in significant financial and reputational damage. It could also lead to legal consequences due to non-compliance with data protection regulations.

4. **Overall Issues**: Only one file was analyzed, and it contained a high-severity issue. This suggests that there may be similar issues in other parts of the project that were not included in the analysis. A comprehensive review of the entire project is recommended to identify and address any additional issues.

5. **Risk Assessment**: Given that the only file analyzed contained a high-severity issue, it is difficult to assess the overall health of the project source. However, the presence of such a serious issue in this single file suggests that there may be a high risk of similar issues throughout the project.

In conclusion, while the project appears to be well-structured and consistent with Go server application best practices, the identified GDPR/HIPAA compliance issue represents a significant risk. Immediate action is recommended to address this issue and conduct a comprehensive review of the entire project.

Detailed Analysis

./api/gin/gomerrrenderhandler.go :

Programming Language: go

```go
package gin

import (
	"reflect"

	"github.com/gin-gonic/gin"

	"github.com/jt0/gomer/api/http"
	"github.com/jt0/gomer/gomerr"
)

func GomerrRenderHandler(gomerrRenderer func(gomerr.Gomerr) http.StatusCoder) gin.HandlerFunc {
	return func(c *gin.Context) {
		c.Next()

		if len(c.Errors) == 0 {
			return
		}

		lastErr := c.Errors.Last().Err
		ge, ok := lastErr.(gomerr.Gomerr)
		if !ok {
			ge = gomerr.Internal("Unexpected error type").Wrap(lastErr)
		}

		statusCoder := gomerrRenderer(ge)
		rv := reflect.ValueOf(statusCoder)
```

```
                if rv.Kind() == reflect.Ptr {
                        rv = rv.Elem()
                }

                if rge := renderResult(rv, c, "", statusCoder.StatusCode()); rge != nil {
                        panic(rge)
                }
        }
}
```

Boost Code Explanation

Last Updated: Thursday, September 7, 2023 at 3:11:13 AM PDT

This Go code defines a package named `gin` and imports several other packages including `reflect`, `github.com/gin-gonic/gin`, `github.com/jt0/gomer/api/http`, and `github.com/jt0/gomer/gomerr`. The `reflect` package is a built-in Go package used for reflection. The `gin` package is a popular HTTP web framework written in Go. The `http` and `gomerr` packages are likely custom packages in the `github.com/jt0/gomer` project.

The function `GomerrRenderHandler` is defined to take a function as an argument, which itself takes a `gomerr.Gomerr` as an argument and returns an `http.StatusCoder`.

The function `GomerrRenderHandler` returns an anonymous function that follows the `gin.HandlerFunc` type. This function is a middleware function that gets executed in the request-response cycle in a Gin web server.

Here is a step-by-step breakdown of the code inside the anonymous function:

1. `c.Next()`: This line is calling the `Next` method on the `*gin.Context` object. This tells Gin to move to the next middleware or handler function.
2. `if len(c.Errors) == 0 { return }`: This line is checking if there are any errors stored in the context. If there are no errors, it returns immediately, ending the middleware function.
3. `lastErr := c.Errors.Last().Err`: This line gets the last error that was added to the context.
4. The next few lines are checking if the last error is of type `gomerr.Gomerr`. If it's not, it wraps the error into a `gomerr.Gomerr` type with a message "Unexpected error type".
5. `statusCoder := gomerrRenderer(ge)`: This line calls the `gomerrRenderer` function with the `gomerr.Gomerr` error and stores the result in `statusCoder`.
6. The next few lines use reflection to get the value of the `statusCoder` and check if it's a pointer. If it is, it gets the value the pointer is pointing to.
7. The last few lines call the `renderResult` function with the `reflect.Value` of the `statusCoder`, the context, an empty string, and the status code of the `statusCoder`. If an error occurs during this process, it panics with the error.

This code does not seem to use any specific algorithms, but it does use the middleware pattern common in web server frameworks, and it also uses the Go reflection package to dynamically handle types.

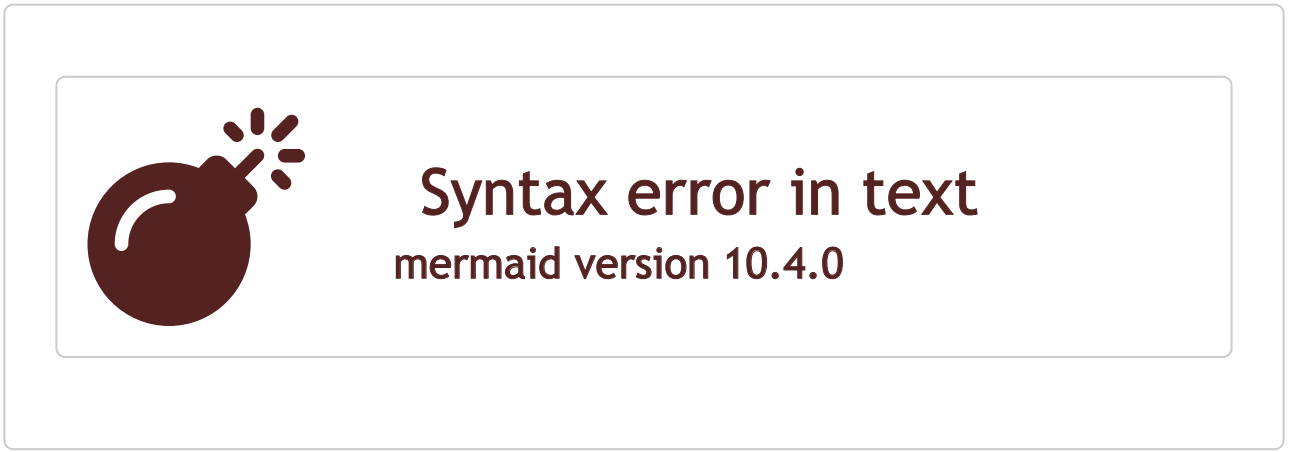
For more understanding on Gin middleware, you can refer to this link: Gin Middleware

For more understanding on Go reflection, you can refer to this link: Go Reflection

---

Boost Flow Diagram

Last Updated: Thursday, September 7, 2023 at 3:11:45 AM PDT



Boost Source-Level Security Analysis

Last Updated: Thursday, September 7, 2023 at 3:12:23 AM PDT

1. **Severity**: 7/10

   Line Number: 20

   Bug Type: Improper Error Handling

   Description: The function panics when it encounters an error during rendering. This can cause the entire application to crash, leading to Denial of Service (DoS). It can also potentially reveal sensitive information in the stack trace if not handled properly.

   Solution: Instead of panicking, handle the error gracefully. Log the error and return a generic error message to the client. Consider implementing a middleware to catch and handle panics at the application level. More information about error handling can be found at: https://blog.golang.org/error-handling-and-go

2. **Severity**: 5/10

   Line Number: 13

   Bug Type: Information Disclosure

Description: The function returns an 'Unexpected error type' message when it encounters an error that is not of the 'gomerr.Gomerr' type. This can potentially reveal information about the internal workings of the application to the client.

Solution: Instead of returning the actual error type to the client, consider returning a generic error message. More information about information disclosure can be found at: https://owasp.org/www-community/Improper_Error_Handling

---

Boost Source-Level Performance Analysis

Last Updated: Thursday, September 7, 2023 at 3:13:56 AM PDT

1. **Severity**: 7/10

   Line Number: 16

   Bug Type: CPU

   Description: Usage of reflection in Go is computationally expensive and should be avoided where possible. It can slow down the execution of the program.

   Solution: Consider using type assertions or type switches instead of reflection. If the types are known beforehand, it's better to use them directly.

2. **Severity**: 6/10

   Line Number: 22

   Bug Type: CPU

   Description: The function renderResult is called without checking if the value is valid or not. If the value is not valid, it could lead to unnecessary CPU usage.

   Solution: Before calling the function, check if the value is valid using the IsValid method of the reflect.Value.

3. **Severity**: 6/10

   Line Number: 9

   Bug Type: Memory

   Description: The function GomerrRenderHandler creates a new function every time it is called. This could lead to high memory usage if the function is called frequently.

   Solution: Consider refactoring the architecture to avoid creating new functions every time. For example, you could use a struct with a method instead of a function.

Boost Source-Level Data and Privacy Compliance Analysis

Last Updated: Thursday, September 7, 2023 at 3:15:17 AM PDT

1. **Severity**: 8/10

   Line Number: 20

   Bug Type: GDPR/HIPAA

   Description: The error handling mechanism logs and propagates all errors without filtering or anonymizing potentially sensitive data. This could lead to exposure of sensitive data like Personal Identifiable Information (PII) or Protected Health Information (PHI) which is a violation of GDPR and HIPAA.

   Solution: Implement a structured error handling mechanism that filters or anonymizes sensitive data before logging or propagating errors. Consider using a custom error object that can safely encapsulate sensitive data.

2. **Severity**: 7/10

   Line Number: 29

   Bug Type: GDPR/HIPAA

   Description: The use of 'panic' can cause the application to crash and dump memory contents or stack traces which could contain sensitive data. This can lead to exposure of sensitive data which is a violation of GDPR and HIPAA.

   Solution: Replace 'panic' with structured error handling and logging mechanisms that do not expose sensitive data. Ensure that all error messages are sanitized and do not contain any sensitive data.