**Objective:**

This document defines a mechanism that enables developers to declare a network error reporting policy for a web application. A user agent can use this policy to report encountered network errors that prevented it from successfully fetching requested resources.   
Well-written applications will dual-purpose logs and activity traces for audit and monitoring, and make it easy to track a transaction without excessive effort or access to the system. They should possess the ability to easily track or identify potential fraud or anomalies end-to-end.

*\*This document is produced by a group of people operating under the supervision of Habib University Faculty\**

**Introduction:**

The worst case scenario is the failure to load the application, or a particular resource, due to a network error, and to address such failures the developer requires assistance from the user agent to identify when, where, and why such failures are occurring. For example, if the user fails to load the page due to a network error, such as a failed DNS lookup, a connection timeout, a reset connection, or other reasons, the site developer is unable to detect and address this issue.

**Error Handling:**

Error handling takes two forms: structured exception handling and functional error checking. Structured exception handling is always preferred as it is easier to cover 100% of code. On the other hand it is very hard to cover 100% of all errors in languages that do not have exceptions, such as PHP 4. Code that covers 100% of errors is extraordinarily verbose and difficult to read, and can contain subtle bugs and errors in the error handling code itself.

* Fail Safe - Applications should always fail safe. If an application fails to an unknown state, it is likely that an attacker may be able to exploit this indeterminate state to access unauthorized functionality, or worse create, modify or destroy data.
* Debug Error – Check if the production code contains bugs or not also do the debug messages leak privacy related information, or information that may lead to further successful attack?
* Exception Handling – that means does the code use structured exception handlers (try () catch () etc) or function-based error handling? Also if the code uses function-based error handling, does it check every return value and handle the error appropriately?
* Funtional Return Values – Refer to the diagram below.

$query = mysql\_query(“SELECT \* FROM table WHERE id=4”, $conn);

if ( $query === false ) {

// error

}

**Key Concepts:**

The key terminologies are:

1. Origin
2. Object and Stringification
3. URL

**Detailed Error Messages:**

Does the browser cache the error messages? And the making sure the production code should not be capable of producing debug messages. If it does, debug mode should be triggered by editing a file or configuration option on the server. In particular, debug should not enabled be an option in the application itself. Detailed error messages, such as stack traces or leaking privacy related information, should never be presented to the user. Instead a generic error message should be used. This includes HTTP status response codes (i.e. 404 or 500 Internal Server error).

**Logging:**

* Where to lop on - Copies of log files should be made at regular intervals depending on volume and size (daily, weekly, monthly, etc.). A common naming convention should be adopted with regards to logs, making them easier to index. Verification that logging is still actively working is overlooked surprisingly often. Log files should be copied and moved to permanent storage and incorporated into the organization's overall backup strategy. Log files and media should be deleted and disposed of properly and incorporated into an organization's shredding or secure media disposal plan. Reports should be generated on a regular basis, including error reporting and anomaly detection trending.
* Handling - All logging components should be synced with a timeserver so that all logging can be consolidated effectively without latency errors. This time server should be hardened and should not provide any other services to the network.
* General Debugging - Logs are useful in reconstructing events after a problem has occurred, security related or not.
* Attack Detection - Logs are often the only record that suspicious behavior is taking place: Therefore logs can sometimes be fed real-time directly into intrusion detection systems
* Quality of Service - Repetitive polls can be protocolled so that network outages or server shutdowns get protocolled and the behavior can either be analyzed later on or a responsible person can take immediate actions.

**How to protect ourselves:**

* Do not allow exceptions to go unhandled
* Do not allow any exceptions to reach the browser
* Display custom error pages to users with an email link for feedback
* Do not enable “Robust Exception Information” in production.