1. Manipulation von Formularen

Formulare können lokal gespeichert und manipuliert werden. Deshalb gilt:

- Attribute wie **`maxlenght=, readonly=, type="number"`** … bieten keinen sicheren Schutz.
- Clientseitige Validierung (z.B.: Javascript) kann umgangen oder deaktiviert werden.
- Vordefinierte Werte (**`checkbox vale="wert"`**) können verändert werden.
- Vordefinierte PHP-Variablen (z.B.: **`$_SERVER["PHP_SELF"]`**) können verändert werden.

> **Serverseitige Datenprüfung (z.B.: mit php) ist zwingend notwendig und geboten!**

2. Prüfung der Variablen

php ist eine schwachtypisierte Sprache. Bei der Übergabe von Daten sind alle Variablen (unabhängig von der Übertragungsart) vom Typ `String`. Einige vordefinierte Funktionen (siehe "php-Grundlagen") werden zum Test bereitgestellt. Die Prüfung der Variablen sollte nach folgenden Regeln (und in dieser Reihenfolge) ablaufen:

2.1. Prüfung auf und Umwandlung in korrekten Datentyp

Funktion / Operatoren	Beschreibung
`bool settype ( $var, string $type )`	Legt den Typ einer Variablen fest (und gibt im Erfolgsfall **`true`** zurück). Mögliche Werte sind: `integer, double, string, array, object.`
Vergleichsoperator **`===`**	Vergleicht Inhalt und Typ einer Variablen auf Identität.
Cast-Operator **`(type)`**`$var`	Wandelt den Typ der Variablen.

2.2. Prüfung auf zulässige Datenlänge und Anzahl

Funktion	Beschreibung
`int strlen ( string $string )`	Liefert die Länge einer Zeichenkette zurück.
`int count ( $var [ , $mode ] )`	Liefert die Anzahl der Elemente innerhalb eines Arrays zurück. Mit `$mode = COUNT_RECURSIVE` wird festgelegt, dass alle Dimensionen gezählt werden.

2.3. Prüfung und Wandlung des Inhaltes

Funktion	Beschreibung
`string strip_tags ($str [ , $tags ] )`	Entfernt innerhalb einer Zeichenkette HTML–Tags. Mit dem optionalen Parameter (tags) können bestimmte HTML-Tags als zulässig deklariert werden.
`string addslashes ( string $str )` (siehe auch Einstellung von **`magic_quotas_gpc`**)	Alle vorkommenden Sonderzeichen (" ' \ ) werden mit einem Backslash geschützt.
`string htmlentities ( string $str )`	Wandelt alle Sonderzeichen und HTML-Tags in den dafür vorgesehenen HTML-Entity-Code.

`string htmlspecialchars ( string $string [, int $flags [, string $encoding ]] )`	Wandelt Sonderzeichen in HTML-Codes.
	Beispiel : Schutz vor XSS durch Anwendereingabe (siehe Ergebnis).
	`$Bsp = '<script>alert("XSS");</script>';`
	`echo htmlspecialchars($Bsp,ENT_QUOTES,'UTF-8');`
	`<script>alert("XSS");</script>`

2.3.1. Bereichs-Prüfung : Prüft, ob Daten innerhalb eines vorgegebenen Bereichs (z.B.: numerisch) liegen.

2.3.2. Whitelist-Prüfung : Prüft auf Gültigkeit der Daten an Hand einer vorgegebenen Elementen – Liste.

2.3.3. Blacklist-Prüfung : Sucht an Hand einer vorgegebenen Liste alle Werte.

> **Als Faustregeln sollten gelten :**
> Grundsätzlich ist eine serverseitige- Validierung einer clientseitigen vorzuziehen!
> Bei festgestellter Manipulation sollte mit Fehlermeldung oder Programmabbruch reagiert werden!

3. Filterfunktionen

Seit PHP 5.2 gibt es Filterfunktionen, welche Variablen und Eingaben prüfen und unschädlich machen. Vor allem wenn Daten aus Formularen stammen oder aus dem Querystring, und bevor Daten in Datenbanken oder Dateien geschrieben oder sonst wie ausgegeben werden sollen, müssen sie zuvor gefiltert werden.

(Quelle : http://www.lehrling.biefer.com/php/filter.php)

Es werden zwei wesentliche Filtervorgänge unterschieden:

1. **Validieren (prüfen),** die Filter heißen FILTER_VALIDATE, und geben nur TRUE oder FALSE zurück.
2. **Bereinigen (säubern, reinigen, heilen)** (FILTER_SANITIZE), sie geben den bereinigten String zurück.

Für jede Filterfunktion gibt es FLAGS und KONSTANTEN:

- Wo kommt die Variable her (INPUT_POST, INPUT_GET, INPUT_COOKIE, INPUT_SERVER, INPUT_ENV) ?
- Wie heißt die Variable bzw. das Eingabefeld ?
- Was soll der Filter tun (VALIDATE oder SANITISE) ?
- Was wird erwartet (STRING, EMAIL, URL oder INT) ?

`print_r(filter_list());`
liefert eine Aufzählung aller PHP-Filter

Funktion	`bool filter_has_var ( int $type , string $var_name )`
Beschreibung	Prüft, ob eine Variable des angegebenen Typs existiert
Argumente	`INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, INPUT_ENV`
Rückgabe	Gibt bei Erfolg **TRUE** zurück. Im Fehlerfall wird **FALSE** zurückgegeben.

Beispiel	`if (false === filter_has_var(INPUT_GET, "email"))`
	`  {echo("Email not found");}`
	`else {echo("Email found");}`

Funktion	`mixed filter_var(mixed var, int $filter [, mixed options])`
Beschreibung	Filtern einer Variablen mit einem angegebenen Filter
Argumente	`FILTER_VALIDATE_* / * => EMAIL, FLOAT, INT, IP, REGEXP, URL` `FILTER_SANITIZE_* / * => STRING, STRIPPED, ENCODED, SPECIAL_CHARS, EMAIL, URL, INT, FLOAT, QUOTES`
Rückgabe	Gibt die gefilterten Daten zurück oder **FALSE** wenn fehlgeschlagen.

Beispiel	`var_dump( filter_var("bob.Beispiel@example.com", FILTER_VALIDATE_EMAIL));`
	liefert `string(24) "bob.Beispiel@example.com"` zurück.
	`var_dump( filter_var("bob.(Beispiel)@example.com", FILTER_VALIDATE_EMAIL));`
	liefert `bool(false)` zurück.
	`var_dump( filter_var("bob.(Beispiel)@example.com", FILTER_SANITIZE_EMAIL));`
	liefert (bereinigt) `string(24) "bob.Beispiel@example.com"` zurück.

Funktion	`mixed filter_input(int $type, string variable, [int filter])`
Beschreibung	Nimmt eine Variable von außen entgegen (und filtert sie optional)
Argumente	Type: `INPUT_GET, INPUT_POST, INPUT_COOKIE, INPUT_SERVER, INPUT_ENV` Filter: `FILTER_SANITIZE_* / * => STRING, STRIPPED, ENCODED, EMAIL, URL, SPECIAL_CHARS, INT, FLOAT, QUOTES`
Rückgabe	Bei Erfolg wird der Wert zurückgegeben, **FALSE** im Fehlerfall oder **NULL**, wenn `string variable` nicht gesetzt ist.

Beispiel	`echo filter_input(INPUT_GET,"email",FILTER_SANITIZE_EMAIL);`
	`$_GET["email"]=>"bob.Beispiel@example.com"` liefert `"bob.Beispiel@example.com"`
	`$_GET["email"]=>"bob.(Beispiel)@example.com"` liefert `"bob.Beispiel@example.com"`
	`vardump(filter_input(INPUT_GET,"email",FILTER_SANITIZE_EMAIL));`
	`$_GET["email"]=>""` liefert `NULL` bzw. `string(0) ""`