Verhinderung von Injection Attacken in Web Applikationen am Beispiel von SMask

BSc. Philipp Lopaur
e9925183@student.tuwien.ac.at
937 9925183

04.12.2007

Zusammenfassung

Diese Arbeit beschreibt eine mögliche Methode Command Injections in Web Applikationen zu verhindern. Der Ansatz lautet SMask [1] und wurde von Martin Johns und Cristian Beyerlein aus Hamburg vorgeschagen. Weiters wird der Ansatz mit SQLCheck [7] einer anderen Methode verglichen.

1 Gründe für Injection Attacken

Die Gründe für Injection Attacken speziell in Web Applikationen sind vielfältig, resultieren aber im allgemeinen daraus dass die Web Applikation strukturierte Daten oder eine andere Sprache für eine Datenbank generiert. Über die Syntax oder den Sinn und Zweck der generierten Sprache hat die Web Applikation aber keine Ahnung. Für die Web Applikation ist der generierte Text eine Blackbox bzw. statischer Text wie für eine GUI Meldung an den Benutzer. Weiters kommt noch hinzu dass statische Blöcke von vorgefertigtem Text mit Daten aus Eingangsparametern manchmal sogar ungeprüft vermischt wird. So kann man über die Eingangsparameter die Syntax der generierten Sprache verändern um z.b.: eine SQL Command Injection durchzuführen.

2 Injection Vulnerbilities in Web Applikationen ein weit verbreitetes Problem

Injection Attacken und speziell SQL Command Injection Attacken sind ein weit verbreitetes Problem. SIe werden laufend auch in grossen Web Applikationen gefunden. Zum Beispiel wurde an einer grossen amerikanischen Universität eine Schwäche bei der Online Studenten Bewerbung entdeckt. Ein Angreifer konnte alle sensiblen Daten wie Lebensläufe, Konto oder Sozialversicherungsnummer aller Studenten durch geeignete Eingangsparamter auslesen. Dieser Vorfall war ein grosser Image Schaden für die Universität da alle hunderttausend Bewerber über den Vorfall informiert werden mussten, damit die Daten nicht mißbräuchlich verwendet werden konnten,

```
// foreign HTML code
echo "<a href='http://www.exa.org'>go</a>";
// foreign SQL code
$sql = "SELECT * FROM users";
$con.execute($sql);
```

Abbildung 1: Fremder Code in der Web Applikation

3 Eigener und fremder Code

Web Applikationen bestehen in der Regel aus mehreren Sprachen. (Abbildung 2) Manche werden am Webserver ausgeführt, manche in der Datenbank und der HTML Output am Ende eines Aufrufes wird schließlich im Web Browser des Benutzers ausgeführt um die entsprechende Web Seite darzustellen. Am wichtigsten ist dabei die Sprache in der die Web Applikation am Server läuft, wie zum Beispiel die interpretierte Sprache PHP oder der JAVA Bytecode der in der JAVA virtuellen Maschine läuft bzw. interpretiert wird. Diese Sprache definieren wir als *nativ*, alle anderen Sprachen mit der die Web Applikation zu tun hat definieren wir als *fremd*

3.1 fremder Code in der Web Applikation

Oft findet man *fremden* Code direkt im Source Code der Web Applikation wie zum Beispiel SQL Satements, oder fragmente von HTML Code. siehe Abbildung 1

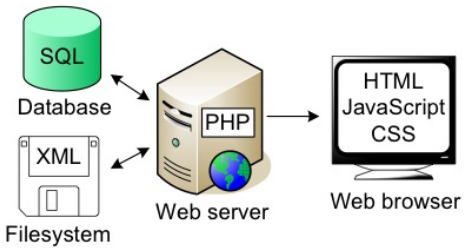

Abbildung 2: Schema einer Web Applikation

4 Arten von Code Injection Attacken

Generell sollte man alle Strings die vom Anwender kommen als pure Daten betrachten und niemals als Code ansehen oder exekutieren. Wenn ein Fehler in der Web Applikation aber dazu führt dass diese Daten ausgeführt werden dann kann ein Angreifer die Kontrolle über das Programm übernehmen. Wir sehen uns folgende Arten von Code Injections an:

4.1 Cross Site Scripting XSS

Hierbei handelt es sich um HTML oder JavaScript Code der in die HTML Ausgabe der Webapplikation gebracht werden soll um beim Betrachter im Browser ausgeführt zu werden um die beabsichtigten Aktionen zu bewirken.

4.2 SQL Injection

Hier sollen die Eingangsparameter der Web Applikation so verändert werden das damit die generierte SQL Sprache zur Abfrage der Datenbank der Web Applikation verändert wird um die beabsichtige Aktion zu bewirken.

4.3 Remote Command Injection

Hier soll ein Programm, Shell oder ähnliches direkt am Webserver gestartet werden. Das geschieht wieder nur durch geschickte Wahl der Eingangsparamter.

5 Das SMask Konzept, Trennung von Daten und Code

Die Idee hinter SMask [1] ist die strikte Trennung von Daten und Code. Wenn niemals Daten von Eingangsparametern oder überhaupt Daten als Code interpretiert und ausgeführt werden dann kann es zu keiner Command Injection kommen.

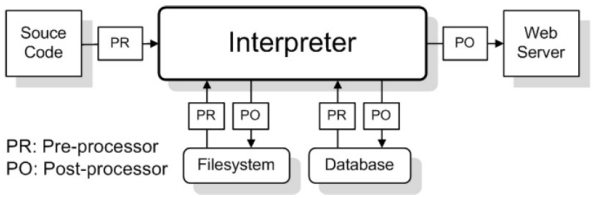

Abbildung 3: Schematische Darstellung von SMask

6 String Masking

In SMask wird die Methode String Masking zur näherungsweisen Trennung von Daten und Code verwendet. Mann kann damit fremden Code ohne Veränderung des String Typs der Sprache der Web Applikation markieren. Dazu wird immer zuerst über den Input der Pre Prozessor und über die Ausgabe der Post Prozessor angewendet, siehe dazu Abbildung 3

6.1 Pre und Post Prozessor

Der Pre Prozessor markiert mithilfe einer Schlüsselwortliste alle Strings in denen fremder Code vorkommt durch zufällige Strings. Der Post Prozessor findet einge-

schleusten Code dadurch das er mithilfe derselben Schlüsselwortliste wieder Schlüsselworte findet. Normalerweise sollten im Text für den Post Prozessor keine Schlüsselwörter mehr vorhanden sein das diese ja vom Pre Prozessor durch andere Strings ersetzt worden sind. Der Post Prozessor neutralisiert eventuell eingeschleusten Code und ersetzt die Maskierungen wieder durch die originalen Schlüsselwörter.

6.2 Maskierung

Die Maskierung von Schlüsselwörtern mit zufalls Strings ist pro Request einmalig und zufällig, wird aber gespeichert um dem Post Prozessor die Umkehrung der Operation zu ermöglichen.

6.3 Schlüsselwortliste

Es wird für die relevante Sprache eine Schlüsselwortliste benötigt. Im Fall von HTML und JavaScript eine liste mit HTML Schlüsselworten und Attributen sowie von JavaScript Funktionen und globaler Variablen.

6.4 Umkehrung der Maskierung

Am Ende nachdem der illegal Code durch entsprechende Encodierung unschädlich gemacht wurde werden die zufalls Strings wieder durch die originalen Schlüsselwörter ersetzt und der Output an den Webserver weitergereicht.

7 False Positives und False Negatives

Es können jeweils beim Pre wie auch beim Post Processor false Negatives sowie false Positives auftreten.

8 Fremden Code teilweise erlauben

Manchmal ist es notwendig Fremden Code doch zu erlauben. Für solche Fläle gibt es in SMask URL basierte Regeln in denen pro URL bestimmte Schlüsselwörter gewhitelisted werden. Zum Beispiel.:

```
<policy unit="post-processor">
<url>/blog/comments.php</url>
<keyword>img</keyword>
<keyword>src</keyword>
</policy>
<policy unit="pre-processor">
<file>/templates/*</file>
<keyword>img</keyword>
...
</policy>
```

9 Mögliche Implementierungen

Smask wurde so ausgelegt dass man ohne Änderung am Programm den Schutz vor Command Injections erhält. Weiters kann Smask auch völlig ohne Veränderung der Interpretes am Webserver nur duch reine automatische Veränderung des Source Codes der Applikations implementiert werden. Die Veränderung der Interpreters durch ein Plugin oder ähnliches macht aber im Bezug auf PHP mehr Sinn da die Plugin Schnittstelle schon existiert und ein Source to Source Prozessor bei der Syntax und dem Programmierstiel der Appliaktionen in PHP sicher sehr problematisch ist.

10 Implementation am Beispiel von PHP

Es wurde SMask anhand von PHP mit einem Plugin [5] für PHP implementiert. Das Plugin hat Zugriff auf alle Eingangsparameter sowie auf den Ausgabebuffer der dem Webserver übergeben wird. Weiters fängt das Plugin alle String Ein- und Ausgabefunktionen in PHP ab. Das Plugin hängt sich auch in den PHP Parser und maskiert alle Strings.

11 Evaluation

Die SMask Implementation wurde mit vielen populären PHP Applikationen wie PHPMyAdmin, PHPNuke, PHPBB, Wordpress oder Tikiwiki getestet und war sehr kompatibel. Es waren keine oder nur minimale Änderungen an den Applikationen notwendig. Es wurden auch verwundbare Versionen von PHPBB getestet wobei SMask alle alten Vulnerbilies abfangen konnte. Es wurde mit [4] getestet.

12 Andere Verfahren

12.1 Manuelle Verfahren

Unter manuell versteht man das Testen und Validieren jedes einzelnen Eingangsparameters. Eine einzelne zentrale Stelle für Regeln ist selten möglich. Es ist daher bei grossen Applikationen unmöglich prinzipielle Sicherheit zu garantieren.

12.2 SQL Injections

Speziell gegen SQL Injections gibt es eine Reihe von Verfahren wie SQLrand oder SQLCheck. SQLrand benutzt eine randomisierte Komponente in jeder SQL Abfrage. Der Nachteil ist das die Applikation und die SQL Abfragen abgeändert werden müssen um SQLrand zu benutzen. Die randomisierung ist ausserdem statisch was bei SQL Fehlermeldungen zu einem Information Leak führen kann und der Angreifer somit die randomisierte Komponente kennt. Zum Verleich arbeitet SMask transparent und verwendet bei jedem Request neue randomisierte Komponenten. SQLCheck [7] verwendet Sprachen in EBNF um veränderte SQL Abfragen noch vor der Datenbank abzufangen. SQL Queries müssen einer definierten Sprache entsprechen.

12.3 Tainting

Tainting markiert alle Eingangsparameter und abgeleitete Variablen als unsicher bis der Programmierer sie explizit als sicher markiert.

12.4 Web Applikation Firewall

Es gibt auch spezielle Web Applikation Firewalls [6] wie Appshield [2] oder Modsecurity [3], die eine gewisse Sicherheit auf der Ebene der Eingangsparameter bringen. So eine Firewall hat allerdings keinen Einblick in die Applikation und kann daher nur als zusätzliche Prüfung der Eingangsparameter betrachtet werden.

13 Bewertung von SMask und SQLCheck

Verglichen mit anderen Verfahren hat SMask den Vorteil das es transparent arbeitet und im Fall der Implementation in PHP sehr kompatibel mit vorhandenen grossen Applikationen ist. Es wurde auch gezeigt das Command Injections verhindert werden. Die zentrale Regeldatei lässt sich auch gut warten. Nachteilig ist die Erstellung einer passenden Schlüsselwortliste, und der generelle Aufwand im Vergleich zur einfachen Encodierung jeglicher Eingangsparameter in unschädliche Strings. Eine prinzipielle Sicherheit kann SMask leider auch nicht garantieren oder beweisen, sodaß je nach Applikation immer noch Command Injections möglich sind, vorzugsweise dort wo gewhitelistet wurde.
SQLCheck geht das Problem gundsätzlicher an der Wurzel an, ist aber auch transparent. Es entsteht jedoch erheblicher Aufwand bei der Definition einer geeigneten Sprache und Regeln genauso wie bei Smask. SQLCheck ist eigentlich nur für strukturierte Sprachen wie SQL geeignet.

Literatur

[1] Martin Johns and Christian Beyerlein. Smask: preventing injection attacks in web applications by approximating automatic data/code separation. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 284–291, New York, NY, USA, 2007. ACM.

[2] Amit Klein. Cross site scripting explained, June 2002.

[3] Ivan Ristic. *Apache Security.* OReilly, March 2005.

[4] RSnake. Xss cheat sheet.

[5] George Schlossnagle. *Advanced PHP Programming.* Sama, February 2004.

[6] David Scott and Richard Sharp. Abstracting application-level web security. In *WWW '02: Proceedings of the 11th international conference on World Wide Web*, pages 396–407, New York, NY, USA, 2002. ACM.

[7] Zhendong Su and Gary Wassermann. The essence of command injection attacks in web applications. In *POPL '06: Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 372–382, New York, NY, USA, 2006. ACM.