Shell-Zugriff per Webbrowser » LinuxCommunity

Originalartikel

Backup

<html>

```
                        <p>    Eine Shell im Browser? PHPshell und Shell in
a Box machen es m&#246;glich und erleichtern damit das Verwalten von
Webservern auch ohne SSH-Zugang &#8211; beispielsweise aus dem n&#228;chsten
Internet-Caf&#233;.</p>
            </div><div class="block" readability="140"><p>
```

Die Verwaltung eines externen Webservers gestaltet sich per Secure Shell einfach, dank X-Forwarding kann man bei entsprechender Netzanbindung sogar grafische Programme zur Verwaltung am heimischen Rechner bedienen. Oft steht aber zur Administration nur ein Rechnern zur Verfügung, auf dem man keine zusätzliche Software installieren kann oder darf. Häufig agiert auch die Firewall so restriktiv, dass außer HTTP(S) nichts hindurch kommt. PHP Shell und Shell in a Box ermöglichen in diesem Fall trotzdem den Shell-Zugriff auf den Server. <h4>PHP Shell</h4> PHP Shell ermöglicht den Shell-Zugang zu Servern, wo die Firewall den Zugang blockiert oder Sie keine Software im Dateisystem installieren können. Ein PHP-fähiger Webserver genügt, um Shell-Befehle auszuführen. Dazu darf der Safe-Mode von PHP nicht aktiviert sein.   Die Installation von PHP Shell funktioniert recht einfach: Sie laden die aktuelle Version von der PHP-Shell-Homepage herunter und entpacken das ZIP-Archiv in ein Verzeichnis auf dem Webspace. Dann setzen Sie ein Passwort, wozu Sie die URL

```
http://<i class="replaceable">Server</i>/phpshell/pwhash.php
```

aufrufen. Dort geben Sie die gewünschte Kombination von Benutzernamen und Passwort ein. Sie erhalten als Ausgabe eine Zeile, die sie im Abschnitt

```
[users]
```

der Konfigurationsdatei

```
config.php
```

eintragen. Bei Bedarf können Sie auch mehrere Benutzer anlegen. Neben Benutzernamen und Passwort lassen sich in

```
config.php
```

noch Shell-Aliases sowie ein Home-Verzeichnis für PHP Shell festlegen. <div class=„box textbox“ id=„article_xtipp“ readability=„62“> Das verschlüsselte Abspeichern des Passworts mithilfe von

```
pwhash.php
```

verhindert zwar unberechtigte Logins, falls einem Angreifer die Konfigurationsdatei in die Hände fallen sollte. Sie sollten PHP Shell aber grundsätzlich über eine per SSL verschlüsselte Verbindung (HTTPS) aufrufen – andernfalls könnte ein Angreifer die eingegebenen Kommandos und Ausgaben im Klartext mitlesen.   Nun steht PHP Shell zum Einsatz bereit. Sie rufen es über die URL

```
https://<i class="replaceable">Server</i>/phpshell/phpshell.php
```

auf, melden sich mit Benutzernamen und Passwort an – und die Shell-Sitzung im Webbrowser kann beginnen. Sie geben nun Kommandos im PHP-Shell-Fenster ein (Abbildung 1). Nach dem Betätigen von [Eingabe] oder einem Klick auf Execute Command werden diese ausgeführt, das Ergebnis erscheint wiederum im Shell-Fenster.

```
<div class="jquerylightbox" readability="12">
    <a
href="http://www.linux-community.de/var/ezwebin_site/storage/images/internal
/artikel/print-artikel/linuxuser/2011/04/shell-zugriff-per-
webbrowser/abbildung-1/1458712-1-ger-DE/Abbildung-1_lightbox.png"
rel="lightbox[image]" title="Abbildung 1: PHP Shell ist unschwer als
Webanwendung zu erkennen.">
    <img
src="http://www.linux-community.de/var/ezwebin_site/storage/images/internal/
artikel/print-artikel/linuxuser/2011/04/shell-zugriff-per-
webbrowser/abbildung-1/1458712-1-ger-DE/Abbildung-1_large.png" alt="" /></a>
    <p>
```

Abbildung 1: PHP Shell ist unschwer als Webanwendung zu erkennen.

```
  </div><p>
```

Die Kommandozeile, die PHP Shell im Browser zur Verfügung stellt, unterliegt dabei einigen Einschränkungen: Jeder Befehl muss ohne weitere Benutzereingabe auskommen, interaktive Programme lassen sich nicht bedienen. Jedes Kommando muss in eine Zeile passen. PHP Shell erkennt nicht, dass ein Kommando noch fortgesetzt werden müsste. So gelingt beispielsweise die Eingabe einer For-Schleife in mehreren Zeilen (wie in der normalen Shell) nicht. Die Befehle müssen innerhalb einer bestimmten Zeitspanne abgearbeitet sein, üblicherweise binnen 30 Sekunden. Hier handelt es sich jedoch nicht um eine durch PHP Shell bedingte Einschränkung: Sowohl der Webserver (meist Apache) als auch PHP brechen nach einer gewissen Zeit die Verarbeitung ab. Sie konfigurieren die entsprechenden Limits in Apache über die

```
Timeout
```

-Direktive, in PHP mittels der Einstellung

```
max_execution_time
```

in

```
php.ini
```

. PHP-Shell führt die Kommandos unter der User- und Group-ID des Webservers aus, wie sich unschwer mit dem

```
id
```

-Kommando überprüfen lässt. Das kann zuweilen recht nützlich sein – etwa, wenn man ein Verzeichnis anlegen möchte, in das nur der Webserver schreiben darf. Das klappt via FTP meist nicht, da man in diesem Fall unter einer anderen User-ID operiert und daher oft lediglich global beschreibbare Verzeichnisse anlegen kann. Mit PHP Shell klappt das dagegen problemlos.   PHP Shell bietet eine einfache History-Funktion, über die Sie mit den Cursortasten in den zuletzt ausgeführten Kommandos vor- und zurückblättern. Umfangreichere History-Funktionen wie eine Suche unterstützt PHP Shell jedoch nicht. Die Größe des „Shell-Fensters“ ändern Sie gegebenenfalls via Size: rechts unterhalb des Eingabebereichs. Dort tragen Sie einfach die gewünschten Werte ein und führen dann den nächsten Befehl aus. PHP Shell beinhaltet auch einen einfachen Editor (

```
editor <i class="replaceable">Datei</i>
```

), mit dessen Hilfe Sie alle Dateien ändern können, auf die der Webserver schreiben zugreifen darf). <h4>Shell in a Box</h4> Shell in a Box (Abbildung 2) eignet sich dann, wenn Sie zwar Shell-Zugang zum Server haben und dort eigene Programme einrichten können, aber als Client einen Webbrowser verwenden wollen oder müssen. Das Projekt stellt neben dem Quellcode auf seinen Webseiten auch DEB-Pakete zur Verfügung. Benutzer von Debian und dessen Derivaten sowie Ubuntu-User installieren die Binärpakete bequem über den Paketmanager. Verwenden Sie eine andere Distribution, entpacken Sie nach dem Herunterladen den Quellcode-Tarball in ein beliebiges Verzeichnis und übersetzen ihn dort mittels

```
./configure ; make
```

. Anschließend installieren Sie Shell in a Box mit dem Befehl

```
make install
```

als Root. Der Aufruf richtet das Programm unterhalb von

```
/usr/local
```

ein. </p>

```
<div class="jquerylightbox" readability="12">
    <a
href="http://www.linux-community.de/var/ezwebin_site/storage/images/internal
/artikel/print-artikel/linuxuser/2011/04/shell-zugriff-per-
webbrowser/abbildung-2/1458719-1-ger-DE/Abbildung-2_lightbox.png"
rel="lightbox[image]" title="Abbildung 2: Shell in a Box wirkt auf den
ersten Blick wie die echte Kommandozeile.">
    <img
src="http://www.linux-community.de/var/ezwebin_site/storage/images/internal/
artikel/print-artikel/linuxuser/2011/04/shell-zugriff-per-
webbrowser/abbildung-2/1458719-1-ger-DE/Abbildung-2_large.png" alt="" /></a>
    <p>
```

Abbildung 2: Shell in a Box wirkt auf den ersten Blick wie die echte Kommandozeile. </p>

```
  </div><p>
```

Shell in a Box bringt anders als PHP Shell einen eigenen Webserver mit, der in der Vorgabe auf Port 4200 lauscht. Dabei kann das Programm eine Reihe von Diensten zur Verfügung stellen, die Sie nach dem Schema </p>

```
  <pre class="auto">shellinaboxd -s <i class="replaceable">Webpfad</i>:<i
class="replaceable">Dienst</i></pre><p>
```

starten. Für erste Versuche können Sie dabei mit der zusätzlichen Option

```
-t
```

oder in der Langform

```
--disable-ssl
```

die Verschlüsselung via SSL (dazu später mehr) vorläufig deaktivieren. Welche grundsätzlichen Möglichkeiten es gibt, führt die Tabelle „Funktionen von Shell in a Box“ auf. Ein

```
shellinaboxd
```

-Prozess kann dabei durchaus mehrere Services zur Verfügung stellen, wie beispielsweise das Login zu mehreren Rechnern: </p>

```
  <pre class="auto">$ shellinaboxd -s /host1/:SSH:host.example.com  -s
/host2/:SSH:host2.example.com</pre><p>
```

So verbinden Sie sich unter der URL

```
http://localhost:4200/host1/
```

zu

```
host.example.com
```

verbinden, unter der URL

```
http://localhost:4200/host2/
```

öffnen Sie eine SSH-Verbindung zum Rechner

```
host2.example.com
```

. <div class=„box table“ readability=„5“>   Funktionen von Shell in a Box   <table class=„renderedtable“ cellpadding=„2“ cellspacing=„0“ id=„article_tfunktionen_von_shell_in_a_box“ readability=„31“><tr><th valign=„top“> Kommando

```
</th><th valign="top">  Funktion
</th></tr><tr class="bglight" readability="2"><td valign="top">
<code>shellinaboxd -s /:LOGIN</code>
</td>
```

<td valign=„top“> Stellt eine Login-Shell am lokalen System unter dem Pfad

```
http://localhost:4200/
```

zur Verfügung.

```
</td>
```

</tr><tr class=„bgdark“ readability=„3“><td valign=„top“>

```
shellinaboxd -s /:SSH
```

```
</td>
```

<td valign=„top“> Stellt ein SSH-Login am lokalen System unter dem Pfad

```
http://localhost:4200/
```

zur Verfügung. Dazu muss ein SSH-Server (zumindest am Loopback-Device) laufen.

```
</td>
```

</tr><tr class=„bglight“ readability=„5“><td valign=„top“>

```
shellinaboxd -s /<i class="replaceable">Bezeichner</i>/:SSH:host.example.com
```

```
</td>
```

<td valign=„top“> Stellt einen SSH-Login für den entfernten Rechner

```
host.example.com
```

am lokalen System unter dem Pfad

```
http://localhost:4200/<i class="replaceable">Bezeichner</i>/
```

zur Verfügung. Dazu muss ein SSH-Server laufen. Auf diese Weise lässt sich Shell in a Box auch als Gateway zu sonst unerreichbaren Rechnern verwenden.

```
</td>
```

</tr><tr class=„bgdark“ readability=„3“><td valign=„top“>

```
shellinaboxd -s /systemstatus/:<i class="replaceable">User</i>:<i
class="replaceable">Gruppe</i>:/:<i class="replaceable">Programm</i>
```

```
</td>
```

<td valign=„top“> Lässt unter

```
http://localhost:4200/systemstatus/
```

ein Programm mit den Rechten des angegebenen Users und der Gruppe laufen. Als Arbeitsverzeichnis dient das Wurzelverzeichnis.

```
</td>
```

</tr></table> Anders als bei PHP Shell (wo eine Zeile komplett eingegeben, diese dann verarbeitet und das Ergebnis an den Browser zurückgeschickt wird), ermöglicht Shell in a Box das textbasierte interaktive Arbeiten, beispielsweise mit einem Editor wie Vi oder Joe. Die Performance bleibt dabei etwas hinter jener im „normalen“ Terminal zurück, meist spürt man jedoch keine großen Einschränkungen.   Shell in a Box stellt noch eine ganze Reihe weitere Optionen zur Verfügung, ein Blick in die Dokumentation beziehungsweise auf die Webseite lohnt sich.   Während erste Experimente mit Shell in a Box am lokalen Rechner durchaus ohne Verschlüsselung erfolgen können, empfiehlt es sich dringend, im produktiven Einsatz alle Verbindungen mittels <a href=„http://www.linux-community.de/Internal/Artikel/Print-Artikel/LinuxUser/2011/04/Shell-Zugriff-per-Webbrowser#article_gssl“ title=„SSL|Secure Sockets Layer. Eine Zwischenschicht, die auf das TCP/IP-

Protokoll aufsetzt und dortexistierende Protokolle (HTTP, POP3, IMAP, …) verschlüsselt.“ target=„_self“ class=„glossary“>SSL abzusichern. Dazu benötigen Sie ein Zertifikat, das sich im entweder im aktuellen Verzeichnis befinden muss oder dessen Speicherort Sie durch die Option

```
--cert=<i class="replaceable">Verzeichnis</i>
```

angeben. Näheres erläutert der Kasten „Verschlüsselung und Zertifikate“.   <div class=„box textbox“ id=„article_xverschl_sselung_und_zertifikate“ readability=„82“>   Verschlüsselung und Zertifikate   Die Verschlüsselung von Passwörtern und Daten zwischen Sender und Empfänger sorgt dafür, dass kein Unberechtigter mitlesen kann. Dazu braucht man Zertifikate, wie man sie von diversen Webseiten (oder auch Mailservern) kennt – meist nimmt man sie erst dann wahr, wenn der Browser ein Zertifikatsproblem meldet. Diese Zertifikate sollen sicherstellen, dass der Server auf der Gegenseite auch tatsächlich derjenige ist, für den er sich dem Client gegenüber ausgibt. Anderenfalls könnte sich ein Angreifer zum Abfangen des Datenverkehrs einfach als berechtigte Gegenstelle ausgeben.   Zertifikate bekommt man einerseits von kommerziellen, kostenpflichtigen Zertifizierungstellen: Sie verifizieren die Identität des Antragstellers und geben dann ein Zertifikat für einen Hostnamen aus. Der Browser erkennt das dann automatisch als korrekt: Er vertraut bestimmten, eingebaute Zertifizierungstellen. Eine Alternative bietet die Community-Lösung CACert .   Schließlich besteht die Möglichkeit, mithilfe von OpenSSL selbst signierte Zertifikate auszustellen – Listing 1 führt die dazu notwendigen Schritte auf. Wenn Sie solche verwenden (Abbildung 3) und von anderen Rechnern aus über das Internet auf derart gesicherte Dienste zugreifen, sollten Sie sich dabei auf jeden Fall das Zertifikat anschauen und die zugehörigen MD5/SHA1-Fingerprints vergleichen.

```
<div class="jquerylightbox" readability="12">
    <a
href="http://www.linux-community.de/var/ezwebin_site/storage/images/internal
/artikel/print-artikel/linuxuser/2011/04/shell-zugriff-per-
webbrowser/abbildung-3/1458726-1-ger-DE/Abbildung-3_lightbox.png"
rel="lightbox[image]" title="Abbildung 3: Ein selbst signiertes Zertifikat
im Webbrowser.">
    <img
src="http://www.linux-community.de/var/ezwebin_site/storage/images/internal/
artikel/print-artikel/linuxuser/2011/04/shell-zugriff-per-
webbrowser/abbildung-3/1458726-1-ger-DE/Abbildung-3_large.png" alt="" /></a>
    <p>
```

Abbildung 3: Ein selbst signiertes Zertifikat im Webbrowser.

```
  </div>
```

<div class=„box listingbox“ id=„article_l1“ readability=„39“>   Listing 1   Key erzeugen:

```
  <pre class="auto"># openssl genrsa -des3 -out server.key 1024</pre><p>
```

Certificate Signing Request (CSR) erzeugen:

```
  <pre class="auto"># openssl req -new -key server.key -out
server.csr</pre><p>
```

Passwort vom Schlüssel entfernen:

```
  <pre class="auto"># cp server.key server.key.org</pre>    <pre
class="auto"># openssl rsa -in server.key.org -out server.key</pre><p>
```

CSR signieren und Zertifikat erzeugen:

```
  <pre class="auto"># openssl x509 -req -days 365 -in server.csr -signkey
server.key -out server.crt
```

cat server.crt server.key > certificate.pem</pre> <div class=„box glossarybox“>   Glossar <dl readability=„5“><dt id=„article_gsafe-mode“> Safe-Mode </dt><dd readability=„6“> Der (als veraltet geltende) Safe-Mode ist ein Versuch, PHP auf Webservern, die viele verschiedene Sites hosten (sogenannte „Shared Server“), durch Einschränkungen gewisser Befehle sicher zu machen. </dd><dt id=„article_gssl“> SSL </dt><dd readability=„6“> Secure Sockets Layer. Eine Zwischenschicht, die auf das TCP/IP-Protokoll aufsetzt und dortexistierende Protokolle (HTTP, POP3, IMAP, …) verschlüsselt. </dd></dl> </html>