Bildquelle: http://www.shutterstock.com/pic.mhtml?id=130266845&src=id

Shellshock - Die Sicherheitslücke auch unter Windows

Einleitung

Shellshock ist die Bezeichnung für eine Familie von Sicherheitslücken in der GNU Bash Shell, dem Kommandozeileninterpreter von unixoiden Betriebssystemen wie beispielsweise Linux. Ursprünglich wurde nur die am 12. September 2014 entdeckte Sicherheitslücke „Bashdoor“ genannt. Nach deren Veröffentlichung und einem ersten Patch am 24. September 2014 wurden weitere Sicherheitslücken identifiziert, welche auf das grundsätzliche Design der Bash und seines Funktionsparsers zurückzuführen sind.

Shellshock ist seit der Entdeckung im Linux/Unix-Bereich relativ bekannt geworden. Weitgehend unbekannt ist dagegen, dass diese Sicherheitslücke in ähnlicher Form auch in der Kommandozeilenumgebung von Microsoft Windows existiert und bis heute von Microsoft nicht geschlossen wurde.

Unsere IT-Experten haben sich mit der Sicherheitslücke näher beschäftigt und fassen für Sie die wichtigsten Punkte zusammen, die es zu beachten gilt. In diesem Artikel soll hauptsächlich die namensgebende Sicherheitslücke CVE-2014-6271 thematisiert werden.

Zunächst wird in diesem Artikel auf die Shellshock-Problematik auf Windows-Systemen und anschließend auf Unix/Linux-Systemen eingegangen.

Einschleusen von Schadcode unter Windows

Der Windows-Kommandozeileninterpreter „CMD.EXE“ gestattet es, beliebige Zeichenketten in scheinbar harmlosen Befehlen einzubetten und ungeprüft auszuführen. Dieses Verhalten beruht nicht auf einem Programmierfehler, sondern wurde im Design des Kommandozeileninterpreters vorgesehen und kann nicht ohne weiteres vom Hersteller geändert werden.

Das Einschleusen von Schadcode in eine Zeichenkette erfolgt dabei mittels Hilfe des Sonderzeichens „&“ („Ampersand“). Dieses wird normalerweise dazu eingesetzt, in einer einzigen Befehlszeile mehrere Befehle hintereinander auszuführen. Es dient dabei als Trennzeichen zwischen den Befehlen, die hintereinander ausgeführt werden sollen. Hier ein Beispiel:

- Syntax: <Befehl1> & <Befehl2> & <Befehl3>
- Beispiel: „**`pause & pause & pause`**“

Wird nun einer beliebigen Zeichenkette ein „&“-Zeichen angehängt, könnte dahinter der Text mit Schadcode angehängt werden:

- Syntax: <Zeichenkette1> & <Schadcodebefehl>
- Beispiel: „**`Willkommen & del c:\docs\*.docx`**“

Wenn diese manipulierte Zeichenkette (inklusive dem Schadcode, der in diesem Beispiel alle Word-Dokumente eines Verzeichnisses löscht) nun einer Variablen zugewiesen werden soll, entsteht das Problem, dass das „&“-Zeichen beim Zuweisen als neuer Befehl angesehen wird. Um dieses Problem zu lösen und das Zuweisen der gesamten Zeichenkette inklusive Schadcode zu ermöglichen, muss die besondere Bedeutung des „&“-Zeichens durch das Caret-Zeichen „^“ aufgehoben werden,

- Beispiel: „**`set variable=Willkommen & del c:\docs\*.docx`**“

Auf diesem Weg kann ausführbarer Code in eine Textvariable eingeschleust werden.

Wird nun der Inhalt der Zeichenkettenvariable innerhalb eines beliebigen CMD-Befehls (z.B. „echo“) ausgewertet, wird neben der Zeichenkette auch der Schadcode mit ausgeführt.

- Beispiel: „**`echo %variable%`**“
- Beispiel: „**`dir %variable%`**“
- Beispiel: „**`copy %variable%`**“

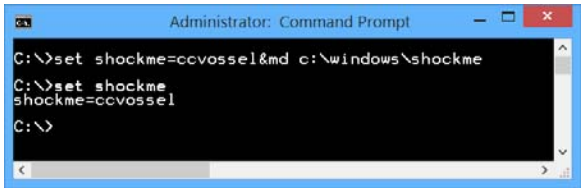
In allen Beispielen wird im Hintergrund der ausführbare Code ausgeführt. Sobald also ein Angreifer es schafft, Schadcode in eine Zeichenkette einzuschleusen, die dann im anzugreifenden System vorhandenen Batch-Skripten verarbeitet wird, wird der Schadcode von diesem Batch-Skript in unbeabsichtigter Weise ausgeführt. Laufen diese Batch-Skripte im administrativen Kontext, so erlangt der Schadcode sogar administrative Rechte und damit die volle Kontrolle des Systems.

Beispiel einer einfachen Schadcode-Einschleusung

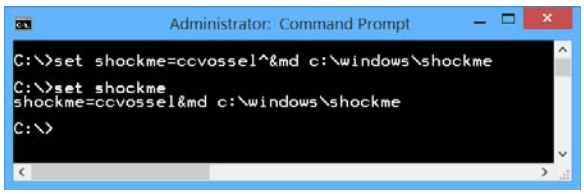
Die folgenden Beispiele dienen zur Erläuterung und können auf jedem modernen Windows-System (z.B. Windows 7, 8, 8.1 oder Windows 10) ohne besondere Hilfsmittel ausprobiert werden. Dazu muss lediglich eine Kommandozeile mit administrativen Rechten unter Windows gestartet werden. Zunächst benötigt man zur Veranschaulichung den administrativen Kontext, der dann später durch Einschleusung in administrative Skripte umgangen werden kann.

In diesem Beispiel soll als Schadcode ein Befehl eingeschleust werden, der innerhalb des geschützten Windows-Verzeichnisses einen Ordner „shockme“ anlegt.

Der Schadcode-Befehl soll daher „md c:\windows\shockme“ lauten. Dieser wird nun in eine herkömmliche Zeichenkette, z.B. „ccvossel“ eingeschleust bzw. durch das „&“-Zeichen angehängt:

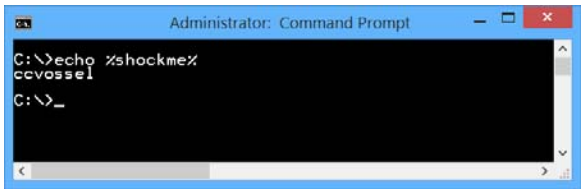


Nach Setzen der Umgebungsvariable und Abfragen deren Wertes erkennt man, dass das Sonderzeichen „&“ und der restliche Text nicht in der Variable gespeichert wurden. Damit das „&“-Zeichen seine besondere Funktion im Kommandointerpreter verliert, muss es mit dem „^“-Zeichen (Caret-Zeichen) entwertet werden:



Nun steht der gesamte Text, inklusive eingeschleustem Schadcode, wie gewünscht in der Variable „shockme“. Soweit zunächst ungefährlich, da hier zwar ein Schadcode innerhalb einer Textvariable gespeichert ist, dieses aber noch nicht ausgeführt wird.

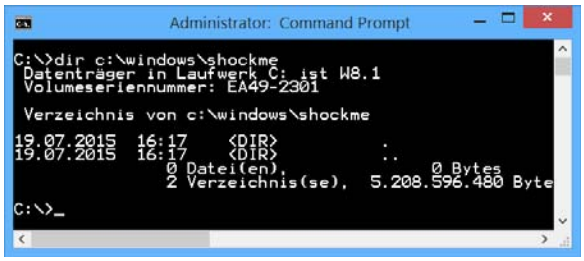
Das Problem tritt bei der Auswertung (Benutzung) der Variable auf, z.B. durch die Ausgabe des Variableninhalts mittel "**echo**":



Es erfolgt nun lediglich die Ausgabe des Textes "ccvossel". Die Ausgabe des Schadcodes wird nicht erzeugt. Der Schadcodeteil wird nicht ausgegeben, sondern einfach ausgeführt.

Das heißt, wenn jemand im administrativen Kontext (ein Administrator oder administrative Batch-Skripte) die Ausgabe einer solchen manipulierten Textvariable generiert, wird zwar der Text „ccvossel“ ausgegeben, aber im Hintergrund Schadcode (hier die Verzeichnisanlage) mit administrativen Rechten ausgeführt.

Die Prüfung zeigt, dass das Verzeichnis angelegt wurde:

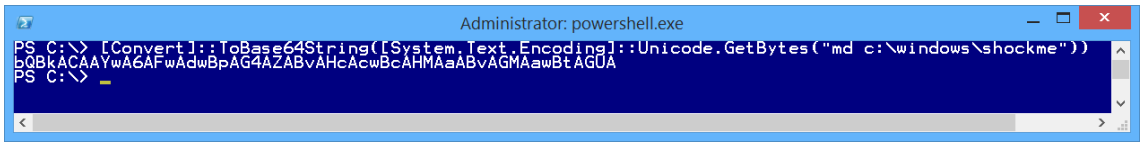


Ebenso ausgeführt wird der eingeschleuste Befehl bei anderen Kommandos, die diese Variable auswerten, z.B. „**`dir %shockme%`**“, „**`type %shockme%`**“ oder „**`copy %shockme%`**“. Das bedeutet, dass alleine durch die Auswertung einer Zeichenkette innerhalb der Shell (z.B. bei einer Ausgabe mit "echo") der enthaltene Schadcode ungeprüft ausgeführt wird. Solche mit Schadcode infizierten Zeichenketten können in Ausgaben von Skripten oder Logfiles enthalten sein oder durch Hacker dort eingeschleust werden. Die eingeschleusten Kommandos können natürlich auch wesentlich mehr Schaden anrichten, als nur ein Verzeichnis in einem administrativen Bereich anzulegen.

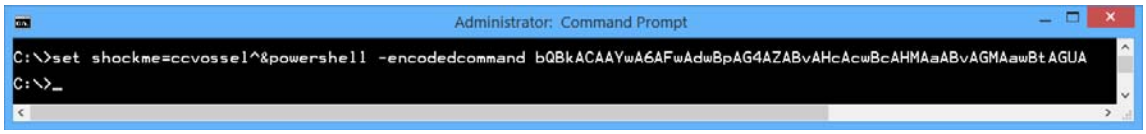
Beispiel einer verschlüsselten Schadcode-Einschleusung

Zudem ist der Einsatz von eingeschleusten Powershell-Befehlen möglich. Diese können sogar Base64-codiert eingeschleust werden, damit die Absicht dieser Befehle nicht im Klartext ersichtlich ist.

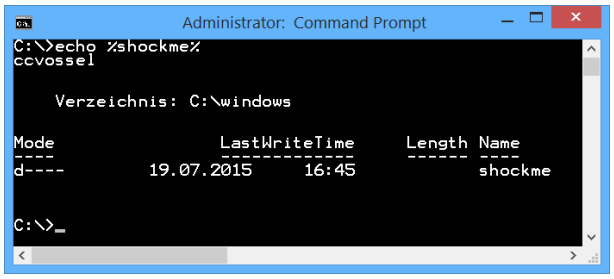
Zunächst wird eine Powershell benutzt, um den oder die Befehle in Base64-Code umzuwandeln:



Dieser Schadcode kann nun in eine Zeichenkettenvariable eingeschleust werden, um es bei der Textausgabe unbemerkt auszuführen zu lassen:



Die Ausgabe der Variable zeigt wieder nur die Zeichenfolge „ccvossel“ an, nicht aber den Schadcode. In diesem Beispiel liefert das Powershell-Kommando eine Ausgabe zurück, die dann ebenfalls ausgegeben wird. Der Schadcode wurde aber auch hier erfolgreich ausgeführt.

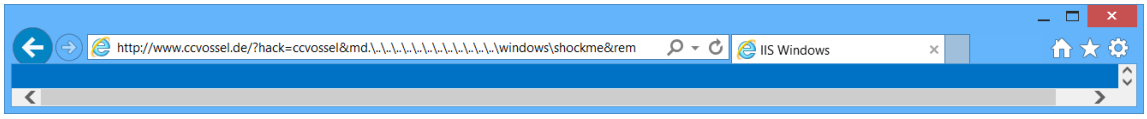


Beispiel einer Schadcode-Einschleusung über einen Webserver

Ein für Hacker interessantes Angriffsszenario ergibt sich, wenn Schadcode über einen öffentlich verfügbaren Webserver einer Firma eingeschleust werden kann. Hier wird dazu ein Windows-Webserver IIS angegriffen.

Dazu kann Schadcode statt wie bisher in Zeichenketten auch in die Aufruf-URLs eines Client-Browsers integriert und beim Webserver eingeschleust werden. In diesem Beispiel wird versucht, Schadcode an einen öffentlichen Webserver zu übertragen. Dazu wird folgende URL eingesetzt:

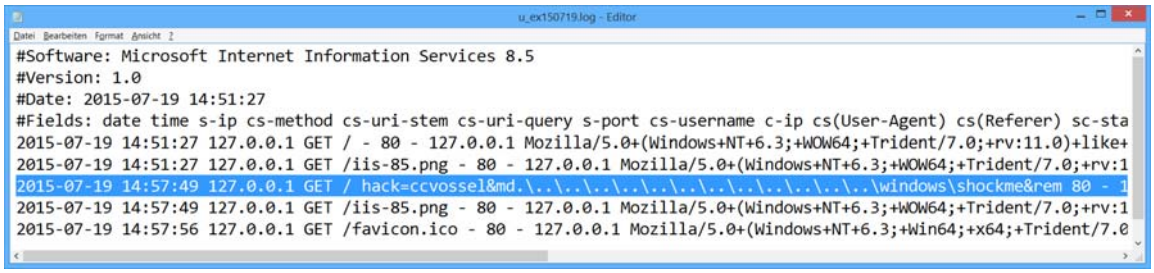
- Beispiel-URL:
  „http://www.ccvossel.de/?hack=ccvossel&md.\..\..\..\..\..\..\..\..\..\..\..\..\windows\shellshock&rem“



Um keine Leerzeichen zu verwenden, die in einer URL zu „%20“ umgewandelt werden würden, wurde das Kommando etwas angepasst. Der mehrfache Einsatz der Zeichenkette „..\“ führt dazu, dass in der Verzeichnisstruktur bis zum obersten Verzeichnis (z.B. „C:“) navigiert wird. Von dort geht es dann ins Windows-Verzeichnis.

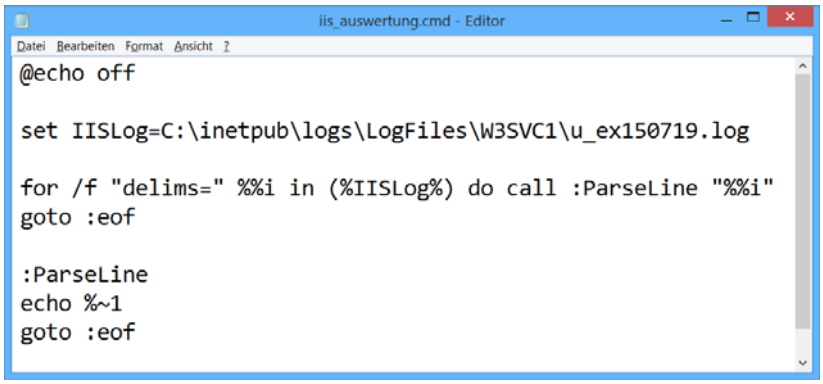
Diese Feinheiten sind bei codierten Powershell-Befehlen nicht zu berücksichtigen. Powershell-Befehle kommen daher hier eher zum Einsatz. Um das Beispiel einfach und lesbar zu halten, wurde hier allerdings darauf verzichtet. Die abschließende Zeichenkette „&rem“ führt dazu, dass weiterer ggf. darauffolgender Text hinter dem Schadcode nicht mehr ausgeführt, sondern als Kommentar ausgewertet wird.

Auf dem Webserver IIS (auf Firmenseite) resultiert daraus folgender Eintrag im Protokoll des IIS-Webservers:

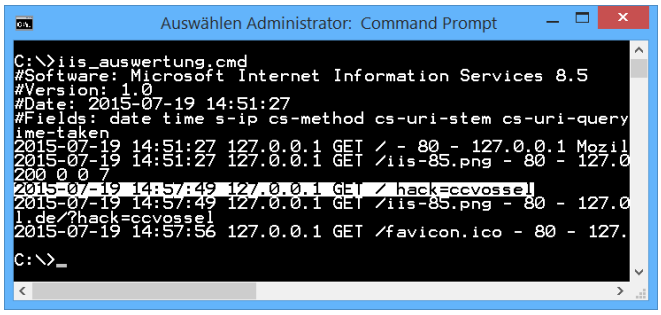


Wird diese Protokolldatei dann manuell durch einen Administrator oder automatisiert innerhalb von administrativen Batch-Skripten analysiert und Zeile für Zeile verarbeitet, können die enthaltenen Schadcode-Befehle unbemerkt ausgeführt werden.

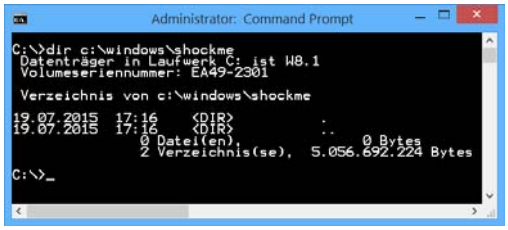
Beispiel für eine Batch-Datei für die automatisierte Protokolldatei-Verarbeitung auf einem Webserver (für diesen Zweck stark vereinfacht):



Durch Ausführen dieser Batch-Datei mit administrativen Rechten wird der eingeschleuste Schadcode dann erfolgreich ausgeführt. In diesem Beispiel wird jede Zeile des Protokolls mittels „echo“ an der Kommandozeile ausgegeben und damit ausgewertet. Enthält eine Zeile präparierten Schadcode, wird dieser ausgeführt.



In diesem Beispiel wurde das Verzeichnis "shockme" erfolgreich und unbemerkt im Windows-Verzeichnis angelegt:



In der Praxis werden Logdateien sehr häufig Zeile für Zeile durch ein Batch-Skript ausgewertet, auch wenn diese normalerweise nicht mit „echo“ ausgegeben werden. Bemerkenswert ist aber, dass jede Art der Auswertung der manipulierten Zeichenketten zur Ausführung des Schadcodes führt, unabhängig ob die Zeichenketten mit „echo“ oder anderen Batch-Befehlen ausgewertet werden.

Einschleusen von Schadcode unter Linux/Unix

Die Bash kann sowohl als eigenständiges Programm als auch als Kommando innerhalb der Bash selbst verstanden werden. Das bedeutet, dass man innerhalb einer Instanz der Bash eine weitere Instanz der Bash aufrufen kann. Die aufrufende Bash vererbt ihre Umgebungsvariablen und Funktionsdefinitionen an die aufgerufene Bash, die sogenannte Kind-Instanz. Umgebungsvariablen werden direkt an die neue Instanz der Shell exportiert. Funktionsdefinitionen hingegen müssen vor der Übergabe die Gestalt einer Umgebungsvariablen annehmen und werden dementsprechend umformatiert. Die neue Instanz der Shell liest bei ihrer Initialisierung die erhaltenen Umgebungsvariablen ein. Funktionsdefinitionen sind dabei durch eine bestimmte Zeichenkette (**() {**) gekennzeichnet, damit sie als Funktionsdefinitionen erkannt und dementsprechend zurückkonvertiert werden können. Nach der Konvertierung werden die Funktionen standardmäßig ausgeführt, ohne den Inhalt der Funktion zu überprüfen.

Wenn man es nun schafft, beliebigen Code in diesen Mechanismus einzuschleusen, kann man diesen ungeprüft ausführen lassen. Dazu setzt man seiner beliebigen Zeichenkette die vom Parser erwartete

Markierung voran und legt sie als Umgebungsvariable ab. Wird nun eine neue Instanz der Shell erstellt, wird dieser beliebige String bei der Übergabe als vermeintliche Funktionsdefinition erkannt und ausgeführt. Von diesem Problem betroffene Shells lassen sich mit folgendem Befehl identifizieren:

- Beispiel: **`env x='() { :;}; echo CVE-2014-6271_in_action' bash -c ""`**

Ist die Bash gefährdet, erscheint „**`CVE-2014-6271_in_action`**“ in der Ausgabe.

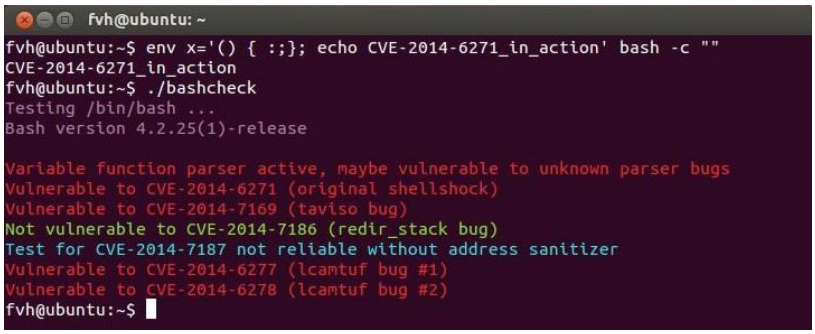


Abbildung: ShellShock: Angreifbare Bash

Ein potentieller Angreifer wird diesen Mechanismus natürlich nicht aushebeln, indem er eine Shell öffnet, seinen Code einschleust und eine neue Instanz der Shell erzeugt. Sollte er in der Lage sein, eine Bash zu starten, dürfte diese eigentlich nicht die Rechte besitzen, Umgebungsvariablen anzulegen oder gar ernsthaften Schaden anzurichten. Wem dies dennoch möglich ist, muss sich nicht mit Shellshock beschäftigen, um Unruhe in fremden Systemen zu stiften.

Der potentielle Schadcode kann im Allgemeinen nicht explizit über die Shell eingebracht werden. Um das System zu gefährden, wird der Code auf Umwegen in die Shell eingeschleust. Der verbreitetste Ansatzpunkt dürfte dabei die Kommunikation des Apache Webservers mit seinen Clients sein. Der Apache reicht Zeichenketten von Clients an untergeordnete Binaries und Skripte weiter. Beim Aufruf dieser CGI- oder PHP-Skripte, die wiederum Bash mit weitreichenden Berechtigungen aufrufen dürfen, könnten manipulierte Umgebungsvariablen, wie $HTTP_COOKIE oder $HTTP_USER_AGENT, den fremden Code in die neue Bash-Instanz exportieren. Dazu muss lediglich der HTTP-Header entsprechend ergänzt und an den Server übertragen werden. Ähnliche Ansätze wurden beispielsweise über die Secure Shell, DHCP oder den Druckerdienst CUPS ausgenutzt.

Betroffen sind zunächst einmal Systeme, welche die GNU Bash bis Version bash43-026 nutzen. Dies sind Unix(-artige) Systeme und Linux.

Achtung: Sechs weitere Sicherheitslücken

Shellshock steht aber auch für sechs weitere Sicherheitslücken, die im Zuge der Analyse von CVE-2014-6271 entdeckt worden sind. CVE-2014-6271 wurde am 24. September 2014, zeitgleich mit einem Fix, veröffentlicht. Die alarmierte Netzgemeinde fand daraufhin weitere Schwachstellen, welche auf diese Sicherheitslücke zurückzuführen sind. Leider hatte der Patch bash43-025 keinen Einfluss auf die neu entdeckten Schwachstellen. Diese wurden aber am 28. September 2014 mit der Version bash43-027 behoben. Sämtliche bekannte Varianten von Shellshock lassen sich mit dem Skript bashcheck [https://github.com/hannob/bashcheck/blob/master/bashcheck] offenlegen. Weitere Gefährdungen, die CVE-2014-6271 zu Grunde liegen, sind nicht bekannt oder es besteht seit Herbst 2014 ein regulärer Fix.

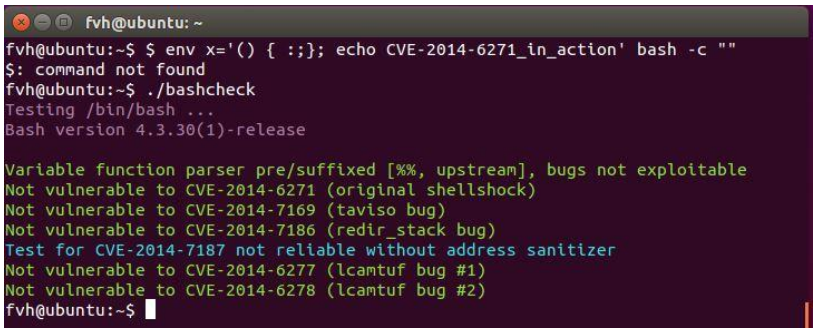


Abbildung: ShellShock: Gefixte Bash

Fazit

Dieser Artikel soll deutlich machen, dass das Problem „Shellshock“ nicht nur auf Unix/Linux-Systeme beschränkt ist, sondern wie bisher kaum bekannt auch in der Windows-Welt von Angreifern ausgenutzt werden kann. Es handelt sich dabei um eine Sicherheitslücke, die nicht ohne weiteres geschlossen werden kann, da es sich um ein Design-Problem der Kommandozeile handelt. Dieser Umstand wird bei Windows-Systemen dadurch verdeutlicht, dass diese Sicherheitslücke bereits vor einem Jahr bekannt wurde und vom Hersteller nicht geschlossen werden konnte. Auch nicht in den Server-Varianten von Windows und in dem vor kurzem erschienenen Betriebssystem Windows 10.

Grundsätzlich sind also alle Windows-Systeme betroffen und angreifbar, wenn auf ihnen Batch-Skripte implementiert sind, die Zeichenketten auswerten, die von Angreifern manipuliert werden können. Dies können, wie im Beispiel gezeigt, Zeichenketten mit Schadcode sein, die ein Angreifer über einen Webserver einschleust und die dann Batch-Skripten ausgewertet werden. Viele weitere Angriffsszenarien sind aber denkbar.

Ein Schutz gegen solche Angriffe ist daher sehr schwierig. Im Prinzip müssen alle Batch-Skripte, die auf den Systemen implementiert sind hingehend der Shellshock-Problematik untersucht und abgesichert werden. Eine Alternative zu diesem Aufwand stellt momentan nur der Verzicht auf Batch-Skripte bzw. deren Ablösung durch beispielweise Powershell-Skripte dar.

Quellen

- Deutsches Forschungsnetz: https://www.dfn-cert.de/aktuell/Shellshock-GNUBash-Schwachstelle-CVE-2014- 6271-CVE-2014-7169-CVE-2014-6277-CVE-2014-7186-CVE-2014-7187.html
- National Vulnerability Database (USA): https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271
- The Security Factory: http://thesecurityfactory.be/command-injection-windows.html
- Projekt-Seite: https://www.trustedsec.com/september-2014/shellshock-dhcp-rce-proof-concept/ https://shellshocker.net/