



GEFÄHRLICHE „GENERALSCHLÜSSEL“ BESEITIGEN DURCH DEN AUFBAU EINER SSH-SCHLÜSSELVERWALTUNG



www.splashtop.com/foxpass



WAS SIND GENERALSCHLÜSSEL?

In vielen Unternehmen erhalten Entwickler uneingeschränkten Zugriff auf jeden Server und alle darauf gespeicherten Daten. Dies wird häufig als Zugriff auf „Administrator“- oder „Root“-Ebene bezeichnet. Ein solch uneingeschränkter Zugriff gilt als „Generalschlüssel“ – so wie der „Generalschlüssel“ eines Bürogebäudes jede Tür öffnet.

Und so wie der Verlust des „Generalschlüssels“ eines Bürogebäudes verheerende Folgen haben kann, kann es noch katastrophaler sein, Entwicklern „Generalschlüssel“ zu Servern und Daten zu geben, wenn einer davon verloren geht, gestohlen oder missbraucht wird.

WARUM SIND GENERALSCHLÜSSEL EIN FEHLER?

FALL 1: BÖSWILLIGER INSIDER – SELTEN, ABER FATAL

Als böswilliger Insider gilt ein Mitarbeiter, der sowohl die nötigen Mittel hat, um auf die Ressourcen eines Unternehmens zuzugreifen, als auch das Motiv, Schaden anzurichten. Dieser Fall ist höchst unwahrscheinlich, da nur Mitarbeiter, die im Unternehmen ein gewisses Maß an Vertrauen gewonnen haben, Root-Zugriff erhalten – ganz ausschließen lässt sich dieses Szenario jedoch nicht.

Ein Beispiel ist der böswillige Insider, der Gigabytes an Daten von Tesla stahl und an Dritte weitergab. Laut einer Studie sind mehr als 49 % der Unternehmen wegen böswilliger Angreifer besorgt, denn dies ist eine reale und wachsende Gefahr.

FALL 2: FAHRLÄSSIGER INSIDER – HÄUFIG UND UNBEMERKT

Trotz aller Schulungen für IT-Fachkräfte sind die meisten Unternehmen nach wie vor anfällig und werden wahrscheinlich Opfer von Phishing und anderen Formen von Social-Engineering-Angriffen. Laut einer Studie wurden 2018 sogar über 34 % aller Sicherheitsverletzungen durch Insider verursacht. Zudem lassen sich solche Angriffe meist nicht nachverfolgen, und die Infrastruktur eines Unternehmens kann bereits kompromittiert sein, bevor jemand bemerkt, was geschehen ist.

FALL 3: GESTOHLENE ZUGANGSDATEN – HÄUFIG UND ERKENNBAR

Gestohlene Laptops oder Zugangsdaten sind ebenfalls wahrscheinliche Angriffe von außen, die auf Mitarbeiter mit den höchsten Zugriffsrechten abzielen, d. h. auf die Entwickler und das interne Team. Sobald ein Außenstehender diese entwendeten Zugangsdaten besitzt, kann er problemlos auf die Infrastruktur des Unternehmens zugreifen. Wurden die Zugangsdaten von mehreren Mitarbeitern gemeinsam genutzt, müssen sie sofort geändert werden – dadurch sind alle anderen Mitarbeiter von den Servern ausgesperrt, bis sie sicher über die neuen Zugangsdaten informiert werden können. Hätte jede Person eigene Zugangsdaten für den Server, müssten nur die der betroffenen Person zurückgesetzt werden.

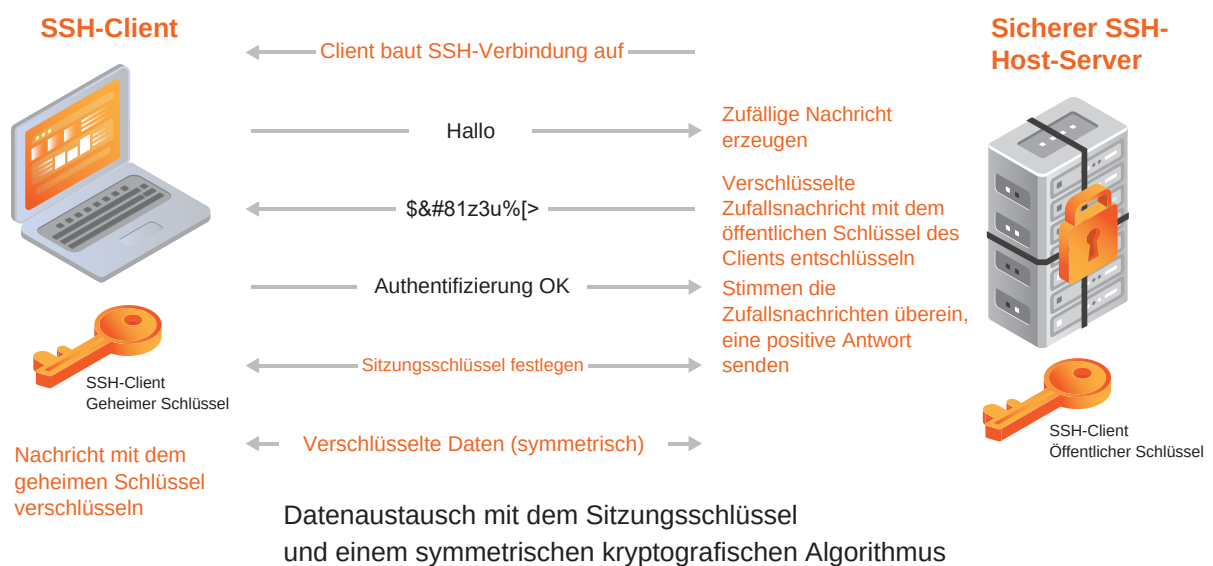


WIE LÄSST SICH DAS RISIKO VON DATENLECKS VERRINGERN?

Technikaffine Unternehmen kennen die Gefahren von Insider-Angriffen und beschränken den Zugriff ihrer Entwickler anhand von Rollen. Die rollenbasierte Zugriffskontrolle (RBAC) begrenzt den Benutzerzugriff auf Basis der jeweiligen Rolle innerhalb einer Organisation.

Beim Einsatz von RBAC hat ein Benutzer nur Zugriff auf das, was er für seine Arbeit häufig benötigt. „Häufig“ muss dabei jedoch nicht unbedingt täglich, wöchentlich oder auch nur monatlich bedeuten. Die sicherste Option besteht darin, Entwicklern die niedrigstmögliche Zugriffsebene zu gewähren. Wird ein höherer Zugriff benötigt, sollte er überwacht, kontrolliert und zeitlich begrenzt sein.

Wie lässt sich ein gut überwachter, kontrollierter und zeitlich begrenzter Zugriffskontrollmechanismus umsetzen? Die beste Lösung ist ein System, in dem jeder Entwickler einen eigenen Schlüssel hat und das den Zugriff auf einen Server für eine vorab festgelegte Dauer erlaubt – je kürzer, desto besser.



WANN SOLLTEN BERECHTIGUNGEN ABLAUFEN?

Normalerweise erhält ein Entwickler Berechtigungen von seinem ersten Tag im Unternehmen bis zu seinem Ausscheiden. Eine dynamischere Option wäre jedoch, Berechtigungen zu gewähren:

- Nur während einer Rufbereitschaft (On-Call-Rotation)
- Bis ein Ticket (z. B. in Jira) geschlossen wird
- Nur wenn die Berechtigung im letzten Monat mindestens einmal genutzt wurde

WAS IST EIN SSH-SCHLÜSSEL?

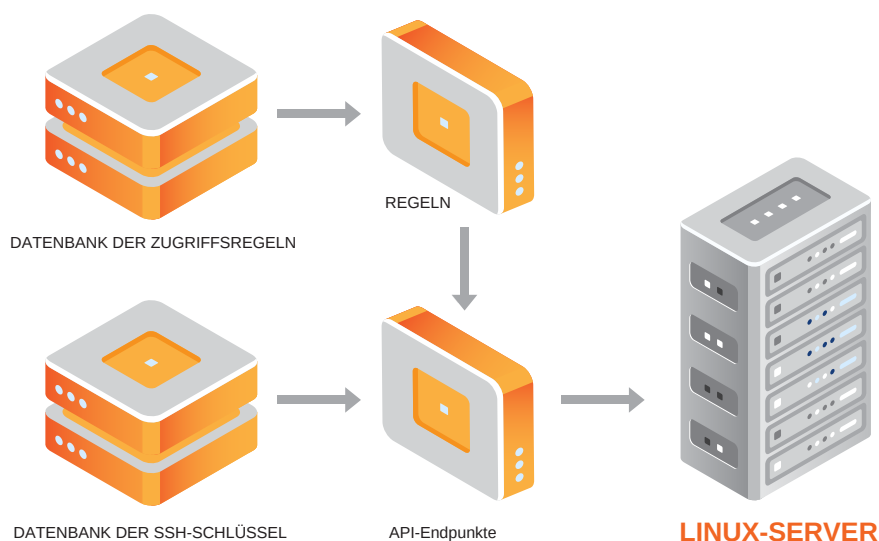
Ein „SSH-Schlüssel“ ist ein kryptografisch sicherer Mechanismus, mit dem man bei der Anmeldung über das Netzwerk seine Identität gegenüber einem Server nachweist. In dieser Hinsicht ähnelt er einem langen Passwort, doch damit enden die Gemeinsamkeiten auch schon. Ein „SSH-Schlüssel“ besteht eigentlich aus zwei Teilen: einem privaten Schlüssel in einer Datei und einem öffentlichen Schlüssel in einer anderen. Die Inhalte dieser beiden Dateien sind sehr unterschiedlich, ergänzen sich jedoch und werden daher gleichzeitig erstellt.

Die private Hälfte des SSH-Schlüssels muss geschützt werden und sollte zum Beispiel niemals das System verlassen, auf dem sie erzeugt wurde. Die öffentliche Hälfte hingegen kann frei weitergegeben werden. Aus dem öffentlichen Schlüssel lässt sich der private Schlüssel nicht rekonstruieren. Wird der öffentliche Teil des Schlüssels auf einen entfernten Server kopiert, auf dem sich dieser Benutzer anmelden darf, erhält der Besitzer der privaten Hälfte Zugriff.

WIE ERSTELLT MAN EIN SYSTEM ZUR SSH-SCHLÜSSELVERWALTUNG?

Für ein System zur SSH-Schlüsselverwaltung werden folgende Komponenten benötigt:

-  1 Eine Datenbank mit den öffentlichen Teilen der SSH-Schlüssel aller Benutzer
-  2 Eine Datenbank mit Regeln, welche Benutzer auf welche Server zugreifen dürfen
-  3 Ein Anwendungsserver für einfache Webanwendungen und API-Endpunkte



SO FUNKTIONIERT DIE EINRICHTUNG:

- 1 Traditionell sucht der sshd-Daemon auf der Festplatte im Verzeichnis ~/.ssh eines Benutzers nach dessen authorized_keys-Datei.

In diesem Setup wird jedoch die sshd-Direktive AuthorizedKeysCommand genutzt (verfügbar ab OpenSSH-Version 6.2), um sshd anzuweisen, die Ausgabe des Skripts unter `~usr/local/bin/get_keys.sh` zu verwenden. Ändern Sie dazu den Pfad in der Datei `sshd_config`. Beachten Sie: Schlägt dies fehl, greift sshd weiterhin auf die `authorized_keys`-Datei auf der Festplatte zurück. Bearbeiten Sie dazu die Datei `sshd_config` wie folgt:

```
ubuntu@host:~$ vi /etc/ssh/sshd_config

[. . .]
AuthorizedKeysCommand /path/to/get_keys.sh
AuthorizedKeysCommandUser nobody
```

- 2 Das Skript unter `~usr/local/bin/get_keys.sh` kann in jeder Sprache geschrieben werden, empfohlen werden jedoch Shell-Skript oder Python. Das erste und einzige Argument dieses Skripts ist der Linux-Benutzername des Benutzers, der die Schlüssel anfordert. Das Skript muss dieselbe Abfrage erneut an den erstellten API-Endpunkt senden. Verwenden Sie dazu den Befehl `curl`, z. B. ``curl https://<endpoint>/get_keys/?user=$1``. Ein Beispiel für `get_keys.sh` (AWS-spezifisch) finden Sie unten:

```
#!/bin/sh

user="$1"
hostname=`hostname`
instance_id=`curl -s -q -f
http://169.254.169.254/latest/meta-data/instance-id`
curl -s -q -f -m 5 "https://keyserver.example.com/sshkeys/?
user=${user}&hostname=${hostname}&instance_id=${instance_id}" 2>/dev/null

exit $?
```

Beispielanfrage:

```
https://keyserver.example.com/sshkeys/?user=aren&hostname=prod-db-1&instance_id=i-0b263919b6498b123
```



Erstellen Sie in der API den Endpunkt /sshkeys/ mit folgenden Funktionen:

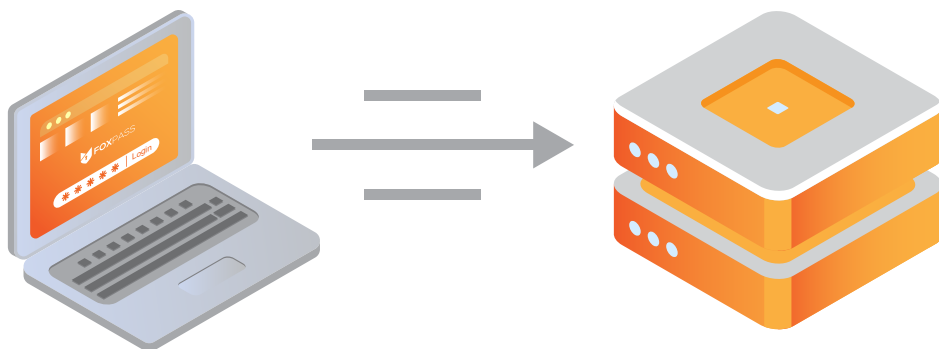
- Er liest das Argument mit dem Benutzernamen aus dem Query-String aus.
- Er sucht die SSH-Schlüssel dieses Benutzers in der Datenbank, und
- Er gibt die SSH-Schlüssel im selben Format wie eine authorized_keys-Datei zurück:
jeden öffentlichen SSH-Schlüssel in einer eigenen Zeile.

Beispiel:

```
def sshkeys(request):  
    hostname = request.GET['hostname']  
    aws_instance_id = request.GET['aws_instance_id']  
    username = request.GET['user']  
  
    user = get_user(username)  
  
    keys = []  
    if is_user_allowed_on_host(user, hostname, aws_instance_id):  
        keys = user.keys  
  
    return HttpResponse("\n".join(key_text) for key in keys),  
    content_type='text/plain')
```



Erstellen Sie eine Webanwendung, mit der Benutzer ihre SSH-Schlüssel verwalten können. Die Authentifizierung eines Benutzers in der Webanwendung lässt sich einfach per OAuth über GSuite oder Office 365 umsetzen. Der Benutzer sollte neue Schlüssel hinzufügen und alte Schlüssel deaktivieren können.



Einfache Web-UI für die Benutzeranmeldung (OAuth)
und das Hochladen der SSH-Schlüssel (öffentlicher Teil)

DATENBANK DER SSH-SCHLÜSSEL

Diese Grundkonfiguration lässt sich um eine Datenbank mit Zugriffsregeln erweitern, die den Zugriff auf die Server regeln und steuern. Durch die Integration externer APIs lässt sich der Zugriff abhängig von Bedingungen anpassen und entsprechend einschränken oder entziehen.

- Damit nur die SSH-Schlüssel aktueller Mitarbeiter verteilt werden: Abgleich mit der API von GSuite, Office365, Okta oder OneLogin
- Für die SSH-Schlüsselrotation: Schlüssel herausfiltern, die älter als 90/180/360 Tage sind
- Um einzuschränken, auf welche Server ein Benutzer zugreifen darf: die für jeden Benutzer definierten Zugriffsrechte prüfen. Besteht für den jeweiligen Server keine Berechtigung, dem Benutzer den Zugriff auf das System verweigern

Beispiel:

<https://keyserver.example.com/sshkeys/?user=aren&hostname=prod-db-1>

```
hostname=prod-db-1
„Produktionsmaschine“ match regex = ^prod-.*
Hat Benutzer „aren“ jetzt Rechte für eine „Produktionsmaschine“?

oder

hostname=prod-db-1
„Datenbankmaschine“ match regex = -db-\d+$
Hat Benutzer „aren“ jetzt Rechte für eine „Datenbankmaschine“?
```

FAZIT

Die Möglichkeiten, dieses Setup für SSH-Schlüsselverwaltung und Zugriffskontrolle zu nutzen, sind nahezu unbegrenzt. Sicherheitslücken und Insider-Angriffe sind für Organisationen mit sensiblen Daten, die mit Entwicklern geteilt werden können, stets ein großes Thema, denn ein Datenleck kann enorme Auswirkungen haben. Es ist immer sinnvoll, Mitarbeitern nur minimale Berechtigungen zu gewähren und den Zugriff auf kritische Ressourcen zu begrenzen, wenn er gewährt wird.

Hätten die Website von Ashley Madison, Target oder Tesla eine bessere Zugriffskontrolle gehabt, wären ihnen solch kostspielige und öffentlichkeitswirksame Sicherheitsvorfälle womöglich erspart geblieben. Die SSH-Schlüsselverwaltung ist ein nützlicher Mechanismus, um Infrastruktur abzusichern und zu verhindern, dass interne Teams Wissen nach außen tragen.

WIE FOXPASS HELFEN KANN

Foxpass bietet benutzerfreundliche, voll ausgestattete Produkte für SSH-Schlüsselverwaltung und privilegierte Zugriffskontrolle, mit denen Entwickler für einen vorab festgelegten Zeitraum auf die benötigten Server zugreifen können. Zudem ist Foxpass ein umfassendes Benutzer- und Gruppenverwaltungssystem für Linux, das sich mit gängigen Verzeichnisdiensten wie GSuite, Office365 und Okta synchronisiert.