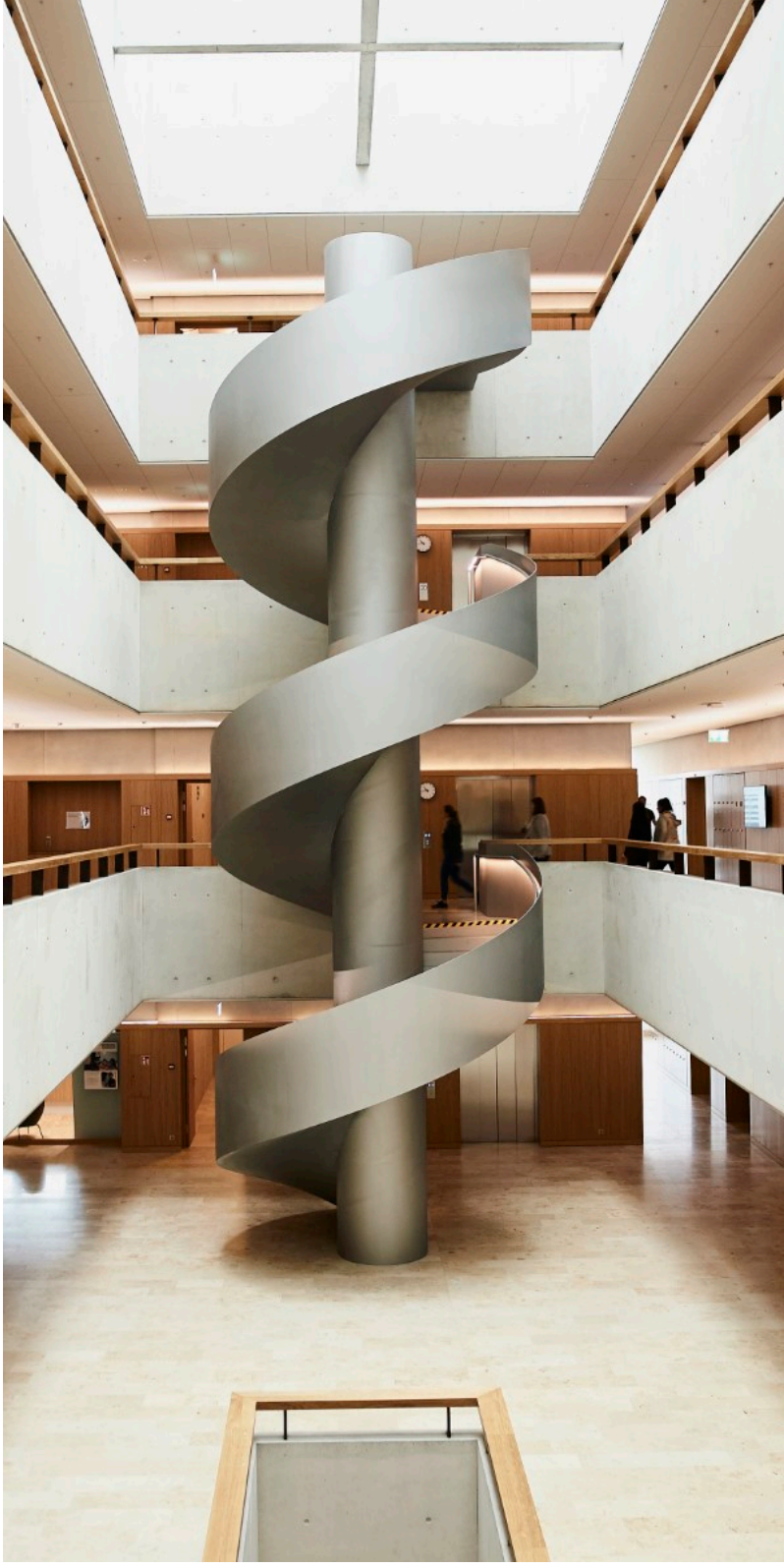
Hochschule Luzern
Modul I.BA_OSSEC



Laborübung

Mandatory Access Control

HSLU Hochschule Luzern

Dokument Version: #03ead27
Template Version: #703559b
Letzte Änderung: 2024-01-29
S. Miescher, J. Zavrlan, M. Useini

I. Allgemeine Informationen

Name:

Gruppe:

Bemerkungen:

Liste der Verfasser

S. Miescher	Creator
J. Zavrlan	Umwandlung in Latex, Korrektur, Feinschliff
M. Useini	Prüflesen, Testen

Copyright Informationen

II. Inhaltsverzeichnis

III. Vorwort

Feedback

Mit Ihrer Mithilfe kann die Qualität des Versuches laufend den Bedürfnissen angepasst und verbessert werden.

Falls in diesem Versuchsablauf etwas nicht so funktioniert wie es beschrieben ist, melden Sie dies bitte direkt dem Laborpersonal oder erwähnen Sie es in Ihrem Laborbericht oder Protokoll. Behandeln Sie die zur Verfügung gestellten Geräte mit der entsprechenden Umsicht.

Bei Problemen wenden Sie sich bitte ebenfalls an das Laborpersonal.

Legende

In den Versuchen gibt es Passagen, die mit den folgenden Boxen markiert sind. Diese sind wie folgt zu verstehen:

Wichtig

Dringend beachten. Was hier steht, unbedingt merken oder ausführen.

Aufgabe III.1

Beantworten und dokumentieren Sie die Antworten im Laborprotokoll.

Hinweis

Ergänzender Hinweis / Notiz / Hilfestellung.

Information

Weiterführende Informationen. Dies sind Informationen, die nicht zur Ausführung der Versuche benötigt werden, aber bekannt sein sollten.

Story

Hierbei wird die Geschichte vermittelt, die in den Versuch einleitet oder den Zweck des Versuches vorstellt.

Zielsetzung

Lernziele, die nach dem Bearbeiten des Kapitels erfüllt sein sollten.

Erkenntnis

Wichtige Erkenntnisse, die aus dem Versuch mitgenommen werden sollten.

1. Vorbereitung

1.1. Tasks

Zur Vorbereitung für die Laborübung sind folgende Tasks im vornherein zu erledigen.

- Lesen Sie Kapitel 2 der Anleitung schon vor dem Labornachmittag durch.

1.2. Benötigte Mittel

Für diesen Versuch wird ein SSH-Zugang zur Laborübungs-VM benötigt.

Sämtliche Aktionen lassen sich über die Kommandozeile durchführen, die VM verfügt nicht über eine grafische Oberfläche.

1.3. Abgabe

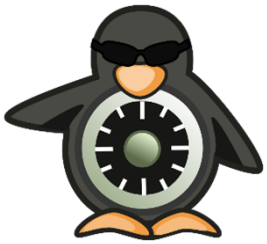
Abgabe des ausgefüllten PDFs auf ILIAS.

2. Linux Kernel Mandatory Access Control

In dieser Laborübung wird AppArmor behandelt, eines von mehreren Security-Modulen des Linux-Kernels, das Mandatory Access Control (MAC) implementiert. AppArmor wurde von Canonical entwickelt und findet primär auf Debian- und Ubuntu-basierten Systemen Anwendung.

Vergleichbare Module sind SELinux von RedHat, welches auf RHEL- und Fedora-basierten Systemen verbreitet ist, sowie TOMOYO von NTT Data Corp, das primär in Japan bekannt ist.

AppArmor-Logo



SELinux-Logo

TOMOYO-Logo

SELinux basiert auf zusätzlichem Tagging im Dateisystem und fügt Rollen für Nutzer und Programme hinzu. Es ist generell komplexer und schwieriger verständlich als AppArmor, welches ein Profil-System für Programme verwendet. TOMOYO verwendet ebenfalls Profile, klassifiziert Programme allerdings in Domains, für welche dann die Profile gelten.

2.1. DAC vs. MAC

Linux-basierte Betriebssysteme haben traditionell ein *Discretionary Access Control* (DAC)-basiertes Sicherheitsmodell. Zugriffe auf Dateien werden anhand von Benutzer- und Gruppenzugehörigkeit sowie der Aktion (lesen, schreiben, ausführen) erlaubt oder verboten. *Mandatory Access Control* (MAC) erweitert dieses Modell und kontrolliert Zugriff für einzelne Programme, meistens im Whitelist-Modus «alles blockieren, was nicht explizit erlaubt wurde» und für zusätzliche Aktionen.

Ausserdem lassen sich mit Linux-MAC-Modulen dateisystemunabhängige Funktionen wie Netzwerkzugriff und Capabilities (privilegierte Kernel-Aktionen) einschränken. Beispielsweise sollte das ‘kill’-Programm in der Lage sein, andere Programme des gleichen Nutzers zu beenden, aber nicht, das System herunterzufahren. Beide Aktionen sind privilegiert, aber nur eine davon ist für das Programm nötig.

2.2. Einsatzzweck von Mandatory Access Control

Mandatory Access Control schränkt Programme darin ein, was diese machen dürfen. Normalerweise kann ein Programm auf alle Dateien zugreifen, zu denen der Benutzer Zugang hat, der das Programm startet. Mit MAC soll das Programm nur noch auf das kleinstmögliche Subset von Daten zugreifen können, das für die Erfüllung seiner Aufgabe zwingend nötig ist.

Wenn alle Applikationen 100% bugfrei sind und immer korrekt konfiguriert würden, bräuchte es selten MAC, weil die Programme dann schon von sich aus nur genau das machen, was sie sollten. Die Hauptaufgabe von MAC ist also primär ein zusätzliches Sicherheitsnetz, das dann greift, wenn irgendetwas falsch läuft.

Der zweite Zweck von MAC ist eine feinere Zugangskontrolle als mit den traditionellen DAC. Sie möchten ein Verzeichnis für die Gruppe «Users» freigeben, **ausser** für den Benutzer «Eve» aus dieser Gruppe? Das geht mit traditionellem

DAC nicht direkt, kann aber mit MAC erreicht werden. Natürlich können Sie mit DAC stattdessen spezifisch eine Gruppe «Users-without-Eve» anlegen.

2.3. AppArmor

AppArmor verwendet ein Profil-System für Programme und folgt einem iterativen Lernprozess. Einem Programm ist generell alles erlaubt, bis ein Profil dafür erstellt und aktiviert wurde. Für einige Programme werden Profile schon mitgeliefert. Der Ablauf beim Erstellen eines neuen Profils für ein Programm ist wie folgt:

1. Erstellen eines leeren AppArmor-Profils für das Programm;
2. Profil in den Überwachungsmodus setzen (Lernprozess startet);
3. Programm starten und normale Programmfunktionalität durchprobieren;
4. Aufgetretene Zugriffe und verwendete Rechte aus dem Überwachungsmodus überprüfen und dem Profil hinzufügen;
5. Schritt 3-4 wiederholen, bis keine neuen Aktivitäten mehr auftauchen;
6. AppArmor-Profil für Programm in «enforce»-Modus setzen (Lernprozess endet, Profil aktiv).

Der Prozess ist also halbautomatisiert, was uns unnötige Arbeit und Fehler erspart, aber trotzdem eine manuelle Kontrolle der Restriktionen ermöglicht.

3. Webapplikation

Nun soll eine Beispiel-Webapplikation abgesichert werden, die aus drei üblichen Teilen besteht:

- Webserver (nginx)
- Serverseitige Programmiersprache (php-fpm)
- Datenbank (mariadb)

3.1. Verhalten ohne AppArmor

Zuerst testen Sie, ob die Webapplikation ohne AppArmor-Konfiguration funktioniert.

3.1.1. Webserver

- Öffnen Sie einen Webbrowser (muss ebenfalls über die VPN-Verbindung laufen)
- Rufen Sie die Test-Website der Labormaschine auf (mit der korrekten Nummer Ihrer VM):

```
1 http://<IP_Adresse_Ihrer_Machine>/
```

Wenn Sie jetzt eine Test-Seite «Welcome to nginx!» sehen, funktioniert der Webserver.

3.1.2. Webapplikation

- Öffnen Sie die Seite der Webapplikation
- Rufen Sie die Test-Website der Labormaschine auf (mit der korrekten Nummer Ihrer VM):

```
1 http://<IP_Adresse_Ihrer_Machine>/index.php
```

Sie sollten jetzt eine Seite «Webapplikation OSSEC» sehen.

Wenn im Abschnitt «Datenbank-Einträge» Daten für «text» und «number» angezeigt werden, funktionieren die Webapplikation und die Datenbankverbindung.

3.1.3. Datei-Upload

- Laden Sie in der Webapplikation eine beliebige Datei hoch.

Erscheint eine grüne Erfolgsnachricht, funktioniert der Datei-Upload.

Falls Sie einen *500 Internal Server Error* erhalten: Bitte beim Laborpersonal melden.

Die Webapplikation hat ein Dateigrössenlimit von 1 MB. Falls Sie ein *413 Request Entity Too Large* bekommen, verwenden Sie eine kleinere Datei.

- Verwenden Sie keine persönlichen Daten zum Testen! Im Zweifelsfall laden Sie ein Bild aus dem Windows-Beispielbilder-Ordner hoch oder erstellen eine triviale Textdatei.

4. AppArmor verwenden

Anhand des nginx-Webservers lernen Sie nun, wie ein AppArmor-Profil erstellt wird.

4.1. Leeres Profil

Erstellen Sie ein neues leeres Profil für nginx:

```
1 sudo aa-autodep nginx
```

Der Befehl aa-autodep generiert ein minimales Profil für ein Programm anhand verlinkter Libraries.

4.2. Lern-Modus

Setzen Sie das Profil in den «complain»-Modus. AppArmor bezeichnet damit den Status eines Profils, bei dem alle Zugriffe und verwendeten Rechte aufgezeichnet werden, aber nicht eingeschränkt sind. Damit lassen sich die notwendigen Informationen sammeln, um das Profil zu trainieren.

```
1 sudo aa-complain nginx
```

4.3. Trainingsdaten sammeln

Nun soll das Programm normal verwendet werden, um herauszufinden, welche Zugriffe der Webserver normalerweise macht.

- Starten Sie den nginx-Service neu:

```
1 sudo service nginx restart
```

- Rufen Sie die **Test-Website** und **die Seite der Webapplikation** nochmals auf (F5 im Browser, oder «Ctrl-L, Enter, F5» für die Webapplikation, damit nicht nochmals eine Datei hochgeladen wird)
- Laden Sie nochmals eine Datei hoch («nochmals senden»-Popup)

4.4. Profil trainieren

Dies ist der essenzielle Teil: Aus den aufgezeichneten Log-Daten sollen nun Einstellungen für das nginx-Profil abgeleitet werden. Wenn das Profil später aktiviert wird, werden jegliche Zugriffe auf Dateien verboten, die im Profil nicht explizit erlaubt sind. Auch Rechte und Netzwerkzugriffe müssen explizit freigegeben werden.

AppArmor stellt für diesen Prozess das Tool «aa-logprof» zur Verfügung:

```
1 sudo aa-logprof
```

4.5. nginx logprof

Es sollte Ihnen bewusst sein, dass unten die Reihenfolge der Berechtigungen nicht immer dieselbe ist. Falls die Berechtigungen in einer anderen Reihenfolgen auftauchen, berücksichtigen Sie sämtliche Anleitungen in diesem Kapitel.

Sie sollten mit dem Befehl sudo aa-logprof eine Ausgabe ähnlich der folgenden sehen:

```
Reading log entries from /var/log/syslog.
Updating AppArmor profiles in /etc/apparmor.d.
Complain-mode changes:

Profile: /usr/sbin/nginx
Capability: dac_override
Severity: 9

 [1 - #include <abstractions/lxc/container-base>]
  2 - #include <abstractions/lxc/start-container>
  3 - capability dac_override,
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

Ihnen stehen nun drei Optionen zur Verfügung.

Die ersten beiden Optionen sind für LXC-Container gedacht. Da wir diesmal nicht mit Containern arbeiten, ignorieren Sie jeweils die LXC-Optionen.

- Drücken Sie die Taste «3» oder zweimal Pfeil-nach-unten, um die dritte Option auszuwählen.

```
[3 - capability dac_override,]
(A)llow / [(D)eny] / (I)gnore / Audi(t) / Abo(r)t / (F)inish
```

- Drücken Sie die Taste «A», um die ausgewählte Option zu erlauben.

Was haben Sie hier grad gemacht? Zuerst hat AppArmor gemerkt, dass nginx eine privilegierte Aktion namens dac_override verwendet hat. Dann haben Sie mehrere Optionen für Berechtigungen, welche diese Aktion erlauben würden. Die ersten beiden Optionen sind sogenannte abstractions, wiederverwendbare Sammlungen von Regeln für bestimmte Zwecke. Wir wollen keinen der beiden abstractions: «container-base» erlaubt generelles Arbeiten mit LXC-Containern, «start-container» erlaubt das Starten von LXC-Containern. Der nginx-Webserver soll aber keine Möglichkeit zur Interaktion mit Containern bekommen.

Stattdessen wählen wir die letzte Möglichkeit, «capability dac_override», um nginx spezifisch die Berechtigung für dac_override zu geben. Mit «Allow» erlauben wir diese Option. Mit «Deny» könnten wir hier auch dac_override explizit verbieten. AppArmor meldet uns, dass das Profil von nginx um die gewählte Berechtigung ergänzt wurde:

```
Adding capability dac_override, to profile.
```

4.5.1. Log-Ordner

Gleich danach wird schon die nächste Berechtigung gefordert:

```
Profile: /usr/sbin/nginx
Path:   /var/log/nginx/error.log
New Mode: w
Severity: 8

 [1 - #include <abstractions/lxc/container-base>]
  2 - #include <abstractions/lxc/start-container>
  3 - /var/log/nginx/error.log w,
(A)llow / [(D)eny] / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew /
   Audi(t) / Abo(r)t / (F)inish
```

Auch hier können Sie die beiden LXC-Optionen ignorieren.

Offenbar möchte nginx auf die Datei */var/log/nginx/error.log* zugreifen, und zwar mit Schreibrechten (das «w» unter «New Mode»). Ein «r» wäre Lesezugriff ohne Schreibrechte.

Anstatt einfach den Zugriff auf die eine Datei zu erlauben, wäre es angebrachter, den Zugriff für alle Dateien im Verzeichnis */var/log/nginx/* freizugeben. Schliesslich ist das Verzeichnis nach dem nginx-Programm benannt.

- Wählen Sie die dritte Option aus.
- Drücken Sie diesmal aber «G» für «Glob»

AppArmor hat uns nun eine vierte Option gegeben, die auch gleich ausgewählt wurde:

```
1 [4 - /var/log/nginx/* w,]
```

- Drücken Sie nun «A», um die vierte Option zu erlauben.

Der Asterisk * im Pfad bedeutet «alle Dateien in diesem Verzeichnis». Sie haben also nginx den Schreibzugriff für alle Dateien in diesem Verzeichnis gegeben. Somit haben Sie sich auch weitere Konfiguration für einzelne Dateien gespart, denn auch */var/log/nginx/access.log* wird von nginx geschrieben.

4.5.2. openssl-Konfiguration

Als nächstes möchte nginx Lese-Zugriff auf die openssl-Konfigurationsdatei. Dieser Zugriff wird für TLS-Verschlüsselung benötigt. Hier ist auch wichtig zu sehen, dass wir explizit nur Lesezugriff erlauben können – wenn nginx Schreibzugriff auf diese Datei anfordert, läuft etwas schief.

```
1  Profile: /usr/sbin/nginx
2  Path:   /etc/ssl/openssl.cnf
3  New Mode: owner r
4  Severity: 2
5
6   [1 - #include <abstractions/lxc/container-base>]
7    2 - #include <abstractions/lxc/start-container>
8    3 - #include <abstractions/openssl>
9    4 - #include <abstractions/ssl_keys>
10   5 - owner /etc/ssl/openssl.cnf r,
```

Diesmal ist eine der gegebenen abstractions tatsächlich hilfreich: openssl gibt uns Zugriff zu dieser Konfigurationsdatei und verwandte Rechte. Mit ssl_keys bekommen wir zusätzlich Zugriff auf sämtliche private keys von openssl, worauf ein Webserver definitiv nicht zugreifen können sollte – Zugriff auf die TLS-Keys für Websites sollte spezifisch einzeln erlaubt werden.

- Wählen Sie die Option «abstractions/openssl» und erlauben Sie diese.

4.5.3. Weitere Berechtigungen

Erlauben Sie nun für die weiteren Berechtigungen selbständig die korrekte Option. Verwenden Sie folgende Einstellungen:

- **/etc/nginx/nginx.conf**: Ganzes Verzeichnis */etc/nginx/* r* erlauben;

- **/etc/nsswitch.conf**: Abstraction «nameservice» erlauben;
- **/etc/nginx/modules-enabled/**: Alle Unterverzeichnisse */etc/nginx/*/ r* erlauben;
- **/usr/share/nginx/modules-available/mod-http-echo.conf**: Gesamten Verzeichnisbaum */usr/share/nginx/*** erlauben (zweimal «Glob» nötig);
- **/var/www/html/index.nginx-debian.html**: Abstraction «web-data» erlauben (siehe unten).

Die vierte Berechtigung mit ** (Verzeichnisbaum-Glob) erlaubt alle Dateien im Verzeichnis sowie alle Unterverzeichnisse und Dateien in Unterverzeichnissen. Diese beinhaltet also auch alles, was in der ersten und dritten Berechtigung freigegeben wurde. Allerdings bekämen wir beim ersten und dritten Punkt bei doppeltem Glob nicht die korrekte Freigabe und müssten manuell nachkorrigieren.

Die letzte Berechtigung mit web-data erscheint nicht immer. Vielleicht haben Sie in Abschnitt 4.3 das Neu-Aufrufen der Test-Website (ohne index.php am Ende) vergessen. Ansonsten können Sie diese Berechtigung manuell hinzufügen, wenn Sie beim Abschnitt 4.9 sind.

4.5.4. Profil speichern

Wenn alle Berechtigungen bearbeitet sind, fragt uns aa-logprof noch, ob die Änderungen am Profil gespeichert werden sollen:

```
= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

 [1 - /usr/sbin/nginx]
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes
   b/w (C)lean profiles / Abo(r)t
```

- Drücken Sie «S», um das Profil zu speichern.

4.6. Prozedur wiederholen

Damit ist das Profil noch nicht fertig: Beim Neustart des nginx-Services verwendet dieser zusätzliche capabilities und Dateizugriffe. Wiederholen Sie also die beiden Schritte so lange, bis aa-logprof keine Änderungen mehr feststellen kann:

- Service neustarten: *sudo service nginx restart*
- Mit *sudo aa-logprof* Berechtigungen hinzufügen

Folgende weitere Berechtigungen sollten Sie dabei verwenden:

- capabilities setuid und setgid
- owner */run/nginx.pid rw*
- owner */etc/nginx/** r* (doppel-Glob nötig)

Es werden Ihnen möglicherweise Abstractions für dovecot und postfix angeboten: Dies sind Mailserver, unser Webserver soll keine Rechte für Mail bekommen. Nginx kann auch für Mailserver-Loadbalancing eingesetzt werden, dann müssen entsprechende Rechte natürlich erlaubt sein. Für den Einsatz als reinen Webserver sind diese aber nicht nötig.

Die nächsten zwei Aktionen können zu einer zusätzlichen Abfrage in aa-logprof führen, je nach Serververhalten und Grösse Ihrer Test-Datei. Falls aa-logprof keine zusätzlichen Rechte anfordert, fahren Sie einfach weiter.

- Rufen Sie die Webapplikation nochmals auf (im Browser F5 drücken)
- Laden Sie eine Datei hoch

Verwenden Sie diese Berechtigung, falls nötig:

- owner */var/lib/nginx/** rw* (doppel-Glob nötig)

4.7. Abstractions

Wenn Sie nun wissen möchten, was genau einzelne vorgeschlagene Abstractions alles machen, können Sie diese im Ordner */etc/apparmor.d/abstractions/* nachschauen.

- Schauen Sie den Inhalt der «nameservice»-Abstraction an:

```
1 cat /etc/apparmor.d/abstractions/nameservice
```

Diese erlaubt ngnix nicht nur einige übliche Dateizugriffe, sondern auch Netzwerk-Zugriff.

4.8. Netzwerkzugriff mit AppArmor

Die Zeilen aus dem letzten Abschnitt sollen nochmals besprochen werden:

```
1   network inet stream,
2   network inet6 stream,
3   network inet dgram,
4   network inet6 dgram,
```

Inet ist IPv4, inet6 ist IPv6, «stream» bezeichnet in der UNIX-Welt das TCP-Protokoll, «dgram» steht für das UDP-Protokoll. TCP stellt den Applikationen ein Interface zur Verfügung, mit dem von einem Byte-Stream gelesen und darauf geschrieben werden kann; wohingegen bei UDP die Daten paketweise (in sogenannten datagrams) zur Verfügung stehen.

Jedenfalls werden mit diesen vier Zeilen alle relevanten Netzwerk-Protokolle erlaubt. Websites via http-Protokoll brauchen zwar nur TCP, aber für DNS-Abfragen muss nginx auch UDP können. Wäre es vielleicht sinnvoll, wenn nur der DNS-Port erlaubt würde für UDP, und nur Port 80 für eingehende Verbindungen? Nein, denn:

- Mandatory Access Control ersetzt keine Firewall.

Die wichtige Aufgabe von AppArmor und anderen MAC-Systemen ist die Möglichkeit, Netzwerkzugang für eine Applikation komplett zu verbieten, wenn eine Applikation keine Netzwerk-Kommunikation braucht. Für feinstufige Netzwerk-Berechtigungen gibt es Firewalls. Da nginx über Internet kommuniziert, sind diese grosszügigen Berechtigungen also korrekt.

4.9. Profil anschauen

Profile sind im Ordner */etc/apparmor.d/*. Öffnen Sie das Profil von nginx in einem Editor, z.B. nano:

```
sudo nano /etc/apparmor.d/usr.sbin.nginx
```

Sie können die beiden überflüssigen Zeilen rauslöschen:

```
  owner /etc/nginx/* r,
  owner /etc/nginx/*/ r,
```

Falls die web-data Abstraction in Abschnitt 4.5.3 gefehlt hat, fügen Sie bei den anderen Abstractions diese Zeile hinzu:

```
  #include <abstractions/web-data>
```

4.10. Zugriffe explizit verbieten

Um Unterbereiche von erlaubten Zugriffen wieder zu blockieren, können explizite Verbots-Regeln definiert werden. Dies machen Sie nun manuell.

- Fügen Sie zuunterst im Profil vor der schliessenden Klammer folgende Zeile hinzu:

```
deny /var/www/html/private/** rw,
```

- Speichern Sie die Änderung am Profil mit Ctrl-O, Enter.
- Beenden Sie den Editor nano mit Ctrl-X.

Im Unterverzeichnis *private/* des Webserver-Contents befinden sich Konfigurationsdateien für die Webapplikation, auf die der Webserver keinen Zugriff haben sollte.

Hinweis

Ein Programm, das ein AppArmor-Profil im Enforce-Modus hat, hat schon per Default alle Zugriffe verboten! Es ist also normalerweise kein explizites Deny auf irgendetwas nötig. Wir brauchen hier nur ein Deny, weil wir ein Verzeichnis für das Programm freigegeben haben (Whitelisting), aber dann ein Unterverzeichnis davon wieder verbieten wollen.

4.11. Fehlkonfigurierter Webserver

Der Webserver wurde falsch konfiguriert: Es wurde vergessen, dass Dateien im Ordner *private/* nicht zugänglich sein sollten! Ein Angreifer könnte so beispielsweise das Datenbankpasswort auslesen.

- Versuchen Sie, das Datenbankpasswort in der Datei dbpassword.txt im Webbrowser anzuschauen:

```
http://<IP_Adresse_Ihrer_Maschine>/private/dbpassword.txt
```

Dafür haben Sie zum Glück mit AppArmor vorgesorgt. Sie müssen nur noch AppArmor mitteilen, dass die Regeln für nginx nun auch angewendet werden sollen.

4.12. Enforce-Modus

Das fertige Profil soll jetzt tatsächlich auch eingesetzt werden. Dafür setzen Sie das Profil vom complain- in den enforce-Modus:

```
1 sudo aa-enforce nginx
```

Zusätzlich sollte nginx noch neu gestartet werden, um den Dienst sauber mit AppArmor gestartet zu haben. AppArmor ist zwar schon aktiv für den laufenden Prozess, es könnte aber sein, dass nginx schon vorher Dateien geöffnet hat, auf die er keinen Zugriff haben sollte.

```
1 sudo service nginx restart
```

Hinweis

Bei Problemen:
Profil zurück in den complain-Modus setzen (sudo aa-complain nginx). Dann schauen, ob zusätzliche Berechtigungen im Logprof auftauchen.

4.13. Profil testen

- Versuchen Sie erneut, das Datenbankpasswort in der Datei dbpassword.txt anzuschauen:

```
1 http://<IP_Adresse_Ihrer_Maschine>/private/dbpassword.txt
```

Jetzt sollte der Zugang auf die Datei über den Webserver nicht mehr funktionieren, eventuell auch hier die Seite neu laden (F5). Damit wurde der Webserver trotz Fehlkonfiguration erfolgreich abgesichert.

Hinweis

Natürlich sollten Sie immer dafür sorgen, dass Dateizugriffsberechtigungen auch im Webserver korrekt konfiguriert sind. Bei Apache2 geschieht dies z.B. über .htaccess-Dateien; bei nginx sollte die Sites-Konfiguration entsprechend angepasst werden.

5. PHP absichern

Der Webserver ist nun mit AppArmor geschützt. Allerdings wird die Webapplikation als PHP-Skript ausgeführt, das zusätzliche Aktionen ausserhalb des Berechtigungssatzes von nginx ausführen kann. Deshalb möchten wir auch den PHP-Dienst absichern. Der PHP-Dienst verwendet das Programm php-fpm7.4 im Ordner */usr/sbin*, welches nicht automatisch erkannt wird, da es nicht im $PATH liegt. Deshalb muss diesmal der ganze Pfad angegeben werden bei den aa-Befehlen:

```
1 sudo aa-autodep /usr/sbin/php-fpm7.4
```

5.1. Basis-Berechtigungen

- Erstellen Sie nun ein Profil für php-fpm analog zur vorherigen Aufgabe.
- Beginnen Sie damit, den php-Dienst mehrfach neu zu starten und daraus Berechtigungen abzuleiten; um die Webapplikation kümmern wir uns später.

```
1 sudo service php7.4-fpm restart
```

- Welche Abstractions geben Sie dem PHP-Dienst?

- Welche Capabilites benötigt der PHP-Dienst?

- Welche Dateien und Verzeichnisse haben Sie explizit freigegeben? Geben Sie auch die Glob-Sterne an und den Modus (r oder w).

- Wiederholen Sie das Neustarten und Hinzufügen von Berechtigungen so lange, bis keine neuen Berechtigungen mehr auftauchen. Ergänzen Sie jeweils die Berechtigungen in den drei vorangehenden Fragen.

5.2. Webapplikation

- Öffnen Sie nun nochmals die Webapplikation neu im Browser (noch nichts hochladen).
- Schauen Sie wieder nach neuen Berechtigungen.
- Welche zusätzliche Abstraction verlangt nun der PHP-Dienst?

- Falls aa-complain keine Änderungen anzeigt, lädt der Browser möglicherweise die Seite nicht neu. Potenziell hilft dann F5, Ctrl-F5 oder Cache leeren und neu laden.
- Laden Sie in der Webapplikation eine Datei hoch (kann die gleiche sein wie schon zuvor).
- Schauen Sie wieder nach neuen Berechtigungen.
- Eine Abstraction «user-tmp» wird jetzt benötigt. Was macht diese (kurz)?

Schliesslich möchte die Webapplikation die Datei noch ins Zielverzeichnis */var/www/files/* kopieren.

- Erlauben Sie den Zugriff für das ganze Verzeichnis */var/www/files/** und speichern Sie das Profil.

5.3. Test: Exploit in der Webapplikation

Auch die Webapplikation hat eine Schwachstelle: Wenn diese mit dem GET-Parameter «exploit» aufgerufen wird, landen die hochgeladenen Dateien nicht in */var/www/files/*, sondern in */var/local/files/*. Dies sollte AppArmor verhindern.

- Öffnen Sie die Webapplikation mit exploit-Parameter

```
1 http://<IP_Adresse_Ihrer_Maschine>/index.php?exploit
```

- Laden Sie wieder eine Datei hoch.

Sie sollten jetzt eine Erfolgsmeldung sehen, dass die Datei ins Verzeichnis */var/local/files/* geladen wurde. Zuvor war jeweils das Verzeichnis in */var/www* und nicht */var/local*.

5.4. Profil aktivieren

- Setzen Sie das Profil für php-fpm in den enforce-Modus.
- Notieren Sie hier den Befehl, den Sie verwenden, um das Profil in den enforce-Modus zu setzen.

- Versuchen Sie den Datei-Upload mit und ohne Exploit nochmals.
- Funktioniert der Upload mit Exploit weiterhin? Funktioniert der Upload ohne Exploit?

6. Was man aus der heutigen Lektion mitnehmen sollte

Sie haben eines der Linux Kernel-based Mandatory Access Control-Systeme kennengelernt und wissen jetzt, wie dieses eingerichtet und verwendet werden kann. Mandatory Access Control setzt das Sicherheitsprinzip «nur so viel Berechtigung wie nötig» in der Praxis um. Whitelist basierende Profile blockieren jegliche Aktionen eines Programms, die nicht explizit erlaubt wurden. Die Beschränkung auf einzelne Programme ermöglicht eine normale Nutzererfahrung mit dem System, indem sie nur jene Prozesse einschränkt, die ein Profil im Enforce-Modus haben.

Wichtig ist der Einsatz von AppArmor bei Services, die aufgrund der Grösse und Komplexität noch unbekannte Bugs enthalten können; bei aus dem Internet erreichbaren Programmen und für Software, deren Konfiguration nicht trivial ist und bei der bei Fehlern unerwünschte Aktionen und Zugriffe möglich werden.

AppArmor ermöglicht die Definition von ‘erwarteten’ Aktionen und Zugriffen eines Programms, entsprechend den Intentionen des Nutzers bzw. Systemadministrators. Es kann effektiv Probleme verhindern, die entstehen, wenn Aktionen technisch möglich, aber nicht intendiert sind. Damit Sie beim Absichern von Webserver etc. nicht bei Null anfangen müssen, gibt es die Möglichkeit, existierende Profile von anderen Leuten zu verwenden. Debian und Ubuntu bringen für einige Software auch Profile mit, diese sind in zusätzlichen Paketen, die Sie installieren können.

Notizen