
Helion
Hartowanie Linuksa we wrogich środowiskach sieciowych
Ochrona serwera od TLS po Tor
Kyle Rankin

Tytuł oryginału: Linux Hardening in Hostile Networks: Server Security from TLS to Tor (Pearson Open Source Software Development Series)

Tłumaczenie: Radosław Meryk

ISBN: 978-83-283-4216-3

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/hartli
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

3

Bezpieczeństwo serwerów

Jeśli ktoś ma zamiar włamać się do serwera, to najbardziej prawdopodobną drogą ataku jest zazwyczaj luka w aplikacji webowej albo innej usłudze bądź hostach serwerów. Napastnik może również dostać się do systemu przez SSH. Sposoby hartowania popularnych aplikacji, dla których serwer może być hostem, omówimy w innych rozdziałach, dlatego w tym skoncentruję się bardziej na ogólnych technikach zabezpieczania niemal dowolnych serwerów — niezależnie od tego, czy jest to host dla serwisu WWW, poczty e-mail, serwisu DNS, czy też czegoś zupełnie innego.

W tym rozdziale omówiono szereg różnych technik hartowania SSH, a także opisano sposób ograniczania szkód, jakie może wyrządzić napastnik lub nawet złośliwy pracownik, jeśli uzyska dostęp do serwera z takimi narzędziami, jak *AppArmor* i *sudo*. Opiszemy także zagadnienia związane z szyfrowaniem dysków w celu ochrony danych w stanie spoczynku, a także sposób konfigurowania zdalnego serwera syslog, aby utrudnić napastnikowi zacieranie śladów.

Część 1. Podstawy bezpieczeństwa serwerów

Zanim przejdziemy do konkretnych metod hartowania, zaczniemy od pewnych podstawowych zasad zabezpieczania serwerów. Przy próbie zabezpieczenia serwera ważne jest, by podchodzić do niego z odpowiednim nastawieniem. Mając to na uwadze, warto pamiętać o kilku zasadach, którymi należy się kierować, niezależnie od serwera, który zabezpieczamy.

Podstawowe praktyki w zakresie zabezpieczania serwerów

Do podstawowych zasad zabezpieczania serwerów należą w szczególności zasada najmniejszych uprawnień, zachowania prostoty oraz dbania o utrzymanie aktualności.

Zasada najmniejszych uprawnień

Zasadę najmniejszych uprawnień będziemy stosowali w całej książce (na przykład w rozdziale 4., „Sieć”, przy okazji omawiania reguł zapór firewall), ale ogólnie rzecz biorąc, powinniśmy dążyć do tego, aby użytkownicy i aplikacje na hoście mieli tylko takie uprawnienia, jakie są konieczne

do wykonania ich pracy, i nic ponadto. Na przykład, konta na serwerach mogą mieć wszyscy programiści, lecz dostęp dla użytkownika *root* powinni otrzymać tylko administratorzy systemów. Można też pójść dalej i przydzielić przeciętnemu programiście dostęp do powłoki środowiska deweloperskiego oraz ograniczyć dostęp do serwerów produkcyjnych tylko dla administratorów systemu i członków personelu pomocniczego, dla których taki dostęp jest niezbędny.

Stosowanie tej zasady do aplikacji oznacza, że aplikacja powinna działać z uprawnieniami użytkownika *root* tylko wtedy, jeśli jest to absolutnie konieczne; w przeciwnym razie powinna ona działać z uprawnieniami jakiegoś innego konta w systemie. Jednym z najczęstszych powodów, dla których aplikacje potrzebują dostępu *roota*, jest otwieranie portu o niskim numerze (otwieranie każdego portu o numerach niższych niż 1025 wymaga uprawnień użytkownika *root*). Dobrym przykładem stosowania tej zasady są serwery WWW. Otwarcie portów 80 i 443 wymaga dostępu *root*, jednak gdy te porty już zostaną otwarte, wtedy przeciętny proces roboczy serwera WWW działa z uprawnieniami mniej uprzywilejowanego użytkownika (np. użytkownika *www-data*). Obecnie powszechnie stosowaną praktyką dla aplikacji webowych jest nasłuchiwanie na portach o wyższych numerach, na przykład 3000 lub 8080. Dzięki temu aplikacje webowe mogą działać z uprawnieniami zwykłego użytkownika.

To ma być proste!

Przy prostym serwerze łatwiej zapewnić bezpieczeństwo niż w przypadku skomplikowanego. Należy unikać instalowania i uruchamiania dodatkowych usług (zwłaszcza usług sieciowych), które nie są potrzebne. Dzięki temu mamy mniej aplikacji, które mogą mieć luki w zabezpieczeniach i dla których należy śledzić dostępność aktualizacji bezpieczeństwa. Twojemu zespołowi oraz audytorom zewnętrznym będzie łatwiej zweryfikować konfigurację, jeśli będziesz utrzymywał pliki danych i wykonywalne w standardowych miejscach i, o ile to możliwe, starał się zachowywać ustawienia domyślne.

Prostota jest również ważna podczas projektowania ogólnej architektury Twojego środowiska. Im bardziej skomplikowana architektura i im więcej „ruchomych części", tym trudniej ją zrozumieć i zabezpieczyć. Jeśli Twój schemat sieci wygląda jak talerz spaghetti, to być może powinieneś rozważyć uproszczenie sposobu komunikowania się serwerów w środowisku. Kiedy dotrzemy do rozdziału o bezpieczeństwie sieci (rozdział 4.), utrzymanie prostoty również ułatwi nam zadanie.

Zachowaj aktualność swoich serwerów

Nowe luki w zabezpieczeniach aplikacji lub bibliotek są znajdowane przez cały czas. Prostym sposobem na pozostanie na szczycie bezpieczeństwa serwerów jest subskrypcja listy mailingowej poświęconej bezpieczeństwu Twojej dystrybucji, a następnie obserwowanie ogłaszanych luk w zabezpieczeniach w celu ustalenia priorytetów aktualizacji serwerów. Im bardziej jednorodne środowisko, tym łatwiej utrzymać aktualność różnych wersji oprogramowania. Zatem będzie nam łatwiej, jeśli będziemy trzymać się jednej dystrybucji Linuksa oraz jej określonej wersji. Listy mailingowe dotyczące bezpieczeństwa dystrybucji Linuksa nie obejmują jednak wykorzystywanego oprogramowania firm trzecich. Z tego powodu warto zapisać się do pomocniczych list dostępnych dla tych produktów.

Konfiguracja SSH

Jedną z najbardziej popularnych usług niemal na każdym serwerze jest SSH. W przeszłości administratorzy używali takich narzędzi, jak telnet, które wysyłały wszystko (w tym hasła!) w formacie zwykłego tekstu, natomiast SSH szyfruje komunikację pomiędzy użytkownikiem a serwerem. Chociaż ta cecha sama w sobie jest usprawnieniem zabezpieczeń, to niestety samo jej stosowanie nie wystarczy. W tej części omówimy kilka podstawowych technik hartowania SSH, które powinniśmy zastosować na wszystkich naszych serwerach.

Wyłącz logowanie na koncie użytkownika root

Jedną z najprostszych czynności, które można wykonać, aby Twoja konfiguracja SSH stała się bardziej bezpieczna, jest wyłączenie logowania na koncie *root*. W dalszej części tego rozdziału opowiemy o tym, jak można uniknąć logowania się z hasłem użytkownika *root* za pomocą narzędzia *sudo* (w niektórych systemach takie podejście jest domyślne), natomiast w tym przypadku chodzi o ograniczenie możliwości logowania się z tożsamością *root* za pomocą hasła, kluczy SSH lub dowolnym innym sposobem. Ze względu na możliwości, jakie ma użytkownik *root*, po prostu bezpieczniej jest wyeliminować możliwość bezpośredniego zalogowania się napastników z uprawnieniami tego użytkownika. Zamiast tego administratorzy powinni logować się jako zwykli użytkownicy, a następnie korzystać z lokalnych narzędzi, takich jak *sudo*, aby stać się użytkownikiem *root*.

Aby wyłączyć logowanie na koncie *root* na serwerze, wystarczy wyedytować plik konfiguracyjny serwera SSH (zwykle w pliku */etc/ssh/sshd_config*) i zmienić ustawienie

```
PermitRootLogin yes
```

na

```
PermitRootLogin no
```

Następnie należy ponownie uruchomić usługę SSH. W zależności od systemu można to zrobić za pomocą jednego z poniższych poleceń:

```
$ sudo service ssh restart
$ sudo service sshd restart
```

Wyłączenie protokołu 1

Stary protokół SSH 1 ma szereg znanych luk w zabezpieczeniach, więc jeśli w Twojej dystrybucji jeszcze go nie wyłączono, trzeba to zrobić. Znajdź wiersz `Protocol` w pliku */etc/ssh/sshd_config* i zadbaj o to, aby miał następujący format:

```
Protocol 2
```

Jeśli byłeś zmuszony do wprowadzenia zmian w pliku, pamiętaj o ponownym uruchomieniu usługi SSH.

Sudo

W dawnych czasach, kiedy administrator musiał wykonać jakieś działania z uprawnieniami użytkownika *root*, logował się bezpośrednio jako użytkownik *root* albo zostawał nim dzięki użyciu takiego narzędzia jak su. Z takim podejściem wiązały się jednak pewne problemy. Zachęcało ono użytkowników do tego, aby pozostawali zalogowani w systemie jako *root*. Ponieważ *root* może zrobić w systemie praktycznie wszystko, co chce, to błędy popełnione jako *root* mogą mieć znacznie gorsze skutki niż w przypadku działań wykonanych z tożsamością zwykłego użytkownika. Ponadto, z punktu widzenia kosztów administracyjnych, w przypadku gdy było kilku administratorów i wszyscy mieli dostęp do serwera z uprawnieniami *root*, trzeba było stworzyć wspólne hasło, znane dla nich wszystkich. Gdy jeden z administratorów opuszczał firmę, pozostali członkowie zespołu musieli zadbać o zmianę haseł dla wszystkich współdzielonych kont.

Narzędzie sudo pomaga rozwiązać wiele z tych problemów dotyczących bezpieczeństwa i zapewnia znacznie silniejszy model zabezpieczeń, który pomaga trzymać się zasady najmniejszych uprawnień. Dzięki sudo administrator może zdefiniować grupy użytkowników, którzy mogą wykonywać zadania z tożsamością innych użytkowników, w tym użytkownika *root*. Narzędzie sudo ma kilka zalet w porównaniu z su:

- Każdy użytkownik wprowadza własne hasło, a nie hasło uprzywilejowanego konta.

  To oznacza, że nie trzeba już zarządzać wspólnymi hasłami do uprzywilejowanych kont. Jeśli użytkownik opuszcza firmę, trzeba tylko wyłączyć jego konto. Oznacza to również, że administratorzy w ogóle nie muszą utrzymywać w systemie haseł dla kont związanych z rolami (w tym użytkownika *root*). Dzięki temu użytkownicy lub cyberprzestępcy w wyniku odgadnięcia hasła nie mogą uzyskać uprawnień, których nie powinni mieć.

- Narzędzie sudo umożliwia szczegółową kontrolę dostępu.

  W przypadku narzędzia su dostęp do uprawnień użytkownika jest działaniem w rodzaju „wszystko albo nic”. Jeśli mogę skorzystać z su, aby uzyskać uprawnienia użytkownika *root*, to mogę w jego imieniu zrobić wszystko, co chcę. Choć oczywiście można utworzyć reguły sudo, które pozwalają na taki sam poziom dostępu, to można również ograniczyć uprawnienia użytkowników tak, aby mogli wykonywać tylko określone polecenia jako *root* lub jako inny użytkownik.

- Narzędzie sudo ułatwia ochronę uprzywilejowanych kont.

  Choć oczywiście można skorzystać z sudo, aby uzyskać kompletny dostęp do powłoki użytkownika *root*, najprostsze wywołanie polecenia sudo to po prostu sudo oraz nazwa polecenia, które ma być uruchomione w imieniu użytkownika *root*. Dzięki temu z łatwością możemy uruchamiać uprzywilejowane polecenia, kiedy jest taka potrzeba, a przez pozostały czas pracować jako zwykły użytkownik.

- Narzędzie sudo zapewnia tzw. ścieżkę audytu (ang. *audit trail*).

  Gdy użytkownik systemu używa sudo, mechanizm audytu śledzi, jaki użytkownik korzystał z sudo, jakie polecenia uruchamiał i kiedy. Rejestrowane są również próby użytkowników skorzystania z sudo w przypadku, gdy nie mają takich uprawnień.

Dzięki temu generowana jest ścieżka audytu, której administrator może później użyć w celu śledzenia prób nieautoryzowanego dostępu do systemu.

Przykłady użycia sudo i najlepsze praktyki

Narzędzie sudo, podobnie jak większość mechanizmów kontroli dostępu, oferuje szeroki zbiór opcji konfiguracyjnych oraz metod grupowania użytkowników, ról i poleceń. Ta konfiguracja zazwyczaj jest zapisana w pliku */etc/sudoers*, chociaż w nowoczesnych systemach często występuje katalog */etc/sudoers.d*, dzięki któremu można lepiej zorganizować specyficzne zestawy reguł sudo w osobnych plikach. Strona podręcznika `man sudoers` (wpisz polecenie `man sudoers`) zawiera wyczerpujące szczegóły na temat budowania własnych złożonych reguł sudo. Dostępnych jest również wiele innych podręczników. Zamiast cytowania tej dokumentacji w tym miejscu opiszę niektóre najlepsze praktyki dotyczące reguł sudo oraz zaprezentuję kilka użytecznych przykładów z nimi związanych. Jednak na początek przyjrzyjmy się uniwersalnemu poleceniu sudo:

```
root ALL=(ALL) ALL
```

To polecenie pozwala użytkownikowi *root* uruchamiać w systemie dowolne polecenie w imieniu dowolnego użytkownika. Pierwsza kolumna określa nazwę użytkownika lub grupy, której dotyczy reguła sudo; w tym przypadku to jest użytkownik *root*. Druga kolumna pozwala określić konkretne hosty, do których ma zastosowanie ta reguła sudo, albo `ALL`, jeśli odnosi się ona do dowolnego hosta. W następnym wpisie, podanym w nawiasach, znajduje się nazwa użytkownika bądź lista użytkowników (oddzielonych przecinkami w przypadku więcej niż jednego użytkownika) — tutaj są to wszyscy użytkownicy. Ostatnia kolumna to rozdzielona przecinkami lista konkretnych programów wykonywalnych w systemie, które można uruchamiać z tymi podwyższonymi uprawnieniami. W tym przypadku są to wszystkie polecenia.

- Użyj *visudo* do edycji pliku */etc/sudoers*.

  Możesz odczuwać pokusę, aby po prostu uruchomić swój ulubiony edytor tekstu i bezpośrednio wyedytować plik */etc/sudoers*. Problem jednak w tym, że jeśli przypadkowo popełnisz literówkę w pliku */etc/sudoers*, możesz całkowicie zablokować sobie dostęp do konta *root*! Narzędzie *visudo* przeprowadza sprawdzanie składni pliku przed jego zapisaniem, dlatego nie ryzykujemy zapisania nieprawidłowego pliku.

- Przydzielaj dostęp grupom użytkowników, a nie konkretnym użytkownikom.

  Stosowanie tej zasady ma większe znaczenie dla wygody administratorów niż dla bezpieczeństwa, ale sudo pozwala na dostęp do grupy w systemie zamiast do konkretnego użytkownika. Oto kilka przykładów reguł sudo, które możesz zastosować w swoim systemie w celu zapewnienia administratorom dostępu z uprawnieniami użytkownika *root*:

  ```
  %admin ALL=(ALL:ALL) ALL
  %wheel ALL=(ALL:ALL) ALL
  %sudo ALL=(ALL:ALL) ALL
  ```

Każda z tych reguł jest równoznaczna z dostępem *root*. Pozwalają one wcielić się w każdego użytkownika w systemie i uruchomić w imieniu tego użytkownika dowolne polecenie. Grupy `admin`, `wheel` i `sudo` to często wykorzystywane grupy w systemie, które mogą być używane w określonej dystrybucji do zdefiniowania tych użytkowników, którzy mogą uzyskać uprawnienia *root*.

Spróbujmy zaprezentować bardziej użyteczny przykład. Załóżmy, że jesteś administratorem pewnej grupy serwerów tomcat, a deweloperzy potrzebują dostępu do lokalnego użytkownika *tomcat* w środowisku deweloperskim, aby mogli rozwiązywać problemy w swoim kodzie. Gdyby na przykład wszyscy użytkownicy w grupie należeli do grupy developers, moglibyśmy dodać do pliku */etc/sudoers* następującą regułę:

```
%developers ALL=(tomcat) ALL
```

- Maksymalnie ograniczaj dostęp do niektórych poleceń.

  Choć oczywiście łatwiej jest po prostu pozwolić komuś uruchamiać wszystkie polecenia jako specyficzny użytkownik, to jeśli chcemy przestrzegać zasady najmniejszych uprawnień, powinniśmy zapewnić użytkownikom dostęp tylko do tych uprzywilejowanych poleceń, które są im potrzebne. Jest to szczególnie ważne w przypadku udzielania dostępu *root*. Na przykład, jeśli administratorzy baz danych (DBA) potrzebują uruchomienia polecenia `psql` jako użytkownicy `postgres`, żeby mieć większą kontrolę nad konfiguracją bazy danych na poziomie systemu, to łatwym sposobem rozwiązania tego problemu jest dodanie następującej reguły:

  ```
  %dbas ALL=(postgres) ALL
  ```

  Problem w tym, że niekoniecznie chcę lub potrzebuję, aby administratorzy DBA robili coś więcej niż uruchamianie `psql`, dlatego mógłbym ograniczyć regułę tylko do tego polecenia, którego administratorzy potrzebują:

  ```
  %dbas ALL=(postgres) /usr/bin/psql
  ```

- Zawsze używaj pełnej ścieżki do skryptów.

  Podczas pisania reguł `sudo` zawsze pamiętaj o podawaniu kompletnej ścieżki do programu wykonywalnego, który użytkownik ma uruchomić. W przeciwnym razie, gdybym po prostu podał `psql` zamiast `/usr/bin/psql`, złośliwy użytkownik mógłby utworzyć lokalny skrypt, nazwać go `psql` i wpisać w nim wszystko, co mu się tylko podoba.

- Twórz skrypty-wrappery w celu ograniczenia poleceń wysokiego ryzyka do specyficznych argumentów.

  W wielu przypadkach, gdy piszesz reguły `sudo`, ostatecznie przydzielasz użytkownikom szersze uprawnienia niż te, których naprawdę potrzebują. Na przykład gdybym chciał umożliwić użytkownikowi zrestartowanie usługi Nginx, mógłbym dać mu dostęp do polecenia `service`:

  ```
  bob ALL=(root) /usr/sbin/service
  ```

  To oczywiście dałoby mu możliwość zrestartowania Nginx, ale jednocześnie także pozwoliłoby mu uruchamiać i zatrzymywać dowolne inne usługi w systemie.

W tych okolicznościach lepiej jest stworzyć niewielki skrypt-wrapper o nazwie */usr/local/bin/restart_nginx* o następującej treści:

```
#!/bin/bash
/usr/sbin/service nginx restart
```

Następnie wystarczy napisać regułę `sudo`, która daje dostęp tylko do tego skryptu:

```
bob ALL=(root) /usr/local/bin/restart_nginx
```

Gdybym chciał zezwolić użytkownikowi `bob` także na zatrzymywanie i uruchamianie usługi Nginx, mógłbym tak zmodyfikować istniejący skrypt, aby akceptował argument wejściowy (który należałoby dokładnie sprawdzić), albo stworzyć dwa nowe skrypty dla funkcji `stop` i `start` w ten sam sposób, jak dla operacji `restart`. W tym drugim przypadku należałoby zaktualizować regułę `sudo` do następującej postaci:

```
bob ALL=(root) /usr/local/bin/restart_nginx, /usr/local/bin/stop_nginx,
/usr/local/bin/start_nginx
```

Należy zadbać o to, aby właścicielem skryptów-wrapperów był tylko użytkownik *root* oraz by tylko *root* miał do nich prawa zapisu (`chmod 775`). Ogólnie rzecz biorąc, należy zachować ostrożność podczas uruchamiania wszelkich skryptów, za pomocą których użytkownik może wykonywać polecenia powłoki (np. `vi`).

- Powstrzymaj się od pisania reguł `sudo` `NOPASSWD`, jeśli nie jest to absolutnie niezbędne.

  Polecenie `sudo` dostarcza flagę o nazwie `NOPASSWD`, która zwalnia użytkownika z obowiązku wprowadzania hasła podczas uruchamiania polecenia `sudo`. W ten sposób można zaoszczędzić trochę czasu, jednak równocześnie eliminujemy jedno z podstawowych zabezpieczeń, jakie gwarantuje polecenie `sudo`, mianowicie konieczność przeprowadzenia uwierzytelniania przez użytkownika, zanim `sudo` pozwoli mu uruchomić polecenie.

  Niemniej jednak istnieją uzasadnione powody korzystania z flagi `NOPASSWD`. W szczególności jest to potrzebne w sytuacji, gdy chcemy uruchomić polecenie z konta określonej roli w systemie, która sama nie ma hasła. Na przykład możesz zezwolić użytkownikowi bazy danych *postgres* na inicjowanie zadania *cron*, które uruchamia z uprawnieniami użytkownika *root* specjalny skrypt tworzący kopię zapasową bazy danych, a konto roli *postgres* nie ma hasła. W takim przypadku należy dodać regułę `sudo` w następującej postaci:

  ```
  postgres ALL=(root) NOPASSWD: /usr/local/bin/backup_databases
  ```

Część 2. Średnio zaawansowane techniki hartowania serwerów

Średnio zaawansowane techniki hartowania serwerów obejmują uwierzytelnianie za pomocą klucza SSH, mechanizm AppArmor i zdalne logowanie.

Uwierzytelnianie za pomocą klucza SSH

Większość administratorów korzysta ze swoich komputerów przez SSH i, niestety, czasami hakerzy też! Jeśli masz serwer udostępniony w publicznym internecie i kiedykolwiek zadałeś sobie trud, żeby sprawdzić logi uwierzytelniania (w systemach opartych na Debianie można je znaleźć w pliku */var/log/auth.log*), możesz się zdziwić, jak wiele prób połączeń przez SSH dociera do Twojej maszyny. Są to ślady ataków siłowych przez SSH. Wielu cyberprzestępców doszło do wniosku, że bardzo często najprostszym sposobem włamania się do serwera Linux jest odgadnięcie hasła użytkownika. Jeśli jeden z użytkowników (lub konto popularnej roli, takie jak *oracle* lub *nagious*) używa hasła, które znajduje się w słowniku napastnika, jest tylko kwestią czasu, kiedy skrypt odgadnie prawidłowe hasło.

Zatem w jaki sposób możemy się bronić przed siłowymi atakami SSH? Jeden ze sposobów to audyt haseł użytkowników i ścisłe egzekwowanie strategii złożoności haseł. Inny może polegać na wybraniu dla SSH innego portu w nadziei na to, że utajnienie portu nas ochroni. Jeszcze inny obejmuje skonfigurowanie systemów, które parsują próby łączenia się przez SSH i modyfikują reguły zapory firewall w przypadku, gdy zbyt wiele prób pochodzi z jednego adresu IP. Oprócz tego, że w ten sposób ryzykujemy zablokowaniem samych siebie, to napastnicy często się „przeprowadzają" i z jednego adresu IP wykonują tylko kilka prób. Następnie przenoszą się na inny host, należący do zazwyczaj ogromnego botnetu. Każda z wymienionych metod może pomóc zmniejszyć ryzyko przeprowadzenia udanego siłowego ataku SSH, ale nie jest w stanie całkowicie go wyeliminować.

Jeśli chcesz całkowicie wykluczyć możliwość przeprowadzania siłowych ataków SSH, to najlepsza metoda jest jednocześnie jedną z najprostszych: należy wyeliminować logowanie do SSH za pomocą haseł. Jeśli usuniesz z równania logowania do SSH za pomocą hasła, to cyberprzestępcy będą mogli odgadywać hasła do woli, a nawet jeśli im się to uda, to i tak nie zdołają się zalogować do SSH.

Jeśli wykluczymy logowanie za pomocą haseł, to jak zdołamy zalogować się do SSH? Najbardziej popularnym zamiennikiem logowania do SSH za pomocą haseł jest używanie par kluczy SSH. Dzięki stosowaniu par kluczy SSH klient (laptop lub inny serwer) ma zarówno klucz publiczny, jak i prywatny. Klucz prywatny jest tajny i przechowywany na Twoim komputerze osobistym, natomiast klucz publiczny jest kopiowany do pliku *~/.plik SSH/authorized_keys* na zdalnym serwerze, na który chcemy się zalogować.

Tworzenie kluczy SSH

Pierwszy krok polega na utworzeniu pary kluczy SSH. Robi się to za pomocą narzędzia *ssh-keygen*. Choć ten program akceptuje wiele różnych opcji i typów kluczy, my skorzystamy z tych, które powinny działać na wielu serwerach:

```
$ ssh-keygen -t rsa -b 4096
```

Opcja `-t` określa typ klucza (`RSA`), natomiast `-b` definiuje rozmiar klucza w bitach (4096 bitów). Taki 4096-bitowy klucz RSA powinien być obecnie akceptowalny. Po uruchomieniu polecenia wyświetli się pytanie o podanie opcjonalnego hasła wymaganego do odblokowania klucza. Jeśli nie wybierzesz hasła, będziesz mógł łączyć się przez SSH do zdalnych serwerów bez konieczności wprowadzania hasła. Wadą takiego podejścia jest to, że jeśli ktoś uzyska

dostęp do klucza prywatnego (domyślnie jest zapisany w pliku *~/.ssh/id_rsa*), to będzie mógł od razu go użyć w celu nawiązania połączenia przez SSH do Twoich serwerów. Radzę ustawić hasło. W jednym z kolejnych punktów opowiem, jak wykorzystać narzędzie *ssh-agent* w celu buforowania hasła w pamięci podręcznej przez określony czas (w podobny sposób hasła `sudo` są często buforowane przez kilka minut, dzięki czemu nie trzeba wpisywać ich wraz z każdym poleceniem).

Po zakończeniu działania polecenia powstaną dwa nowe pliki: klucz prywatny w pliku *~/.ssh/id_rsa* oraz klucz publiczny w pliku *~/.ssh/id_rsa.pub*. Klucz publiczny można bezpieczne współdzielić z innymi osobami. Jest on zapisany w pliku, który będzie skopiowany na zdalne serwery. Z kolei klucz prywatny powinien być chroniony tak samo jak hasła lub inne tajne informacje i nie powinien być współdzielony z innymi.

Można rozważyć opcję tworzenia różnych kluczy SSH do różnych celów. Podobnie jak posługiwanie się różnymi hasłami do różnych kont, korzystanie z różnych kluczy SSH dla różnych kont jest zgodne z zasadą kompartmentalizacji i pomaga chronić nasze zasoby w sytuacji, gdy jeden z kluczy zostanie skradziony. Aby przechowywać parę kluczy w pliku z inną nazwą niż domyślna, należy użyć opcji `-f`, która pozwala wskazać inny plik. Na przykład, jeśli używasz tego samego komputera do użytku osobistego i do pracy, właściwe będzie stworzenie oddzielnej pary kluczy dla każdego środowiska:

```
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/workkey
```

W wyniku wykonania powyższego polecenia w katalogu *~/.ssh/* zostaną utworzone pliki *workkey* i *workkey.pub*.

Kopiowanie kluczy SSH na inne hosty

Po wygenerowaniu kluczy SSH należy skopiować zawartość klucza publicznego do pliku *~/.ssh/authorized_keys* na zdalnym serwerze. Chociaż można połączyć się przez SSH ze zdalnym serwerem i zrobić to ręcznie, łatwiej to zrobić za pomocą narzędzia o nazwie *ssh-copy-id*. Na przykład, jeśli chcę skopiować klucz publiczny na serwer o nazwie *web1.example.com*, a mój login to *kyle*, mogę wpisać następujące polecenie:

```
$ ssh-copy-id kyle@web1.example.com
```

Należy zastąpić nazwę użytkownika i nazwę serwera nazwą użytkownika i serwera, których będziesz używać do zalogowania się na zdalny serwer. W tym momencie wyświetli się pytanie o hasło potrzebne do zalogowania na zdalnym komputerze, ale kiedy polecenie zostanie wykonane, będzie to ostatni raz! Podobnie jak w przypadku zwykłego logowania przez SSH, jeśli Twoja lokalna nazwa użytkownika jest taka sama, jak nazwa użytkownika na zdalnym serwerze, możesz pominąć ją w poleceniu. Polecenie `ssh-copy-id` domyślnie skopiuje plik *id_rsa.pub*, ale jeśli para kluczy ma inną nazwę, to możesz użyć argumentu `-i`, aby podać inny klucz publiczny. Gdybyśmy więc chcieli użyć niestandardowego pliku *workkey*, który stworzyliśmy wcześniej, napisalibyśmy:

```
$ ssh-copy-id -i ~/.ssh/workkey.pub kyle@web1.example.com
```

Przydatną własnością polecenia `ssh-copy-id` jest to, że dba ono o ustawienie odpowiednich uprawnień do katalogu *~/.ssh*, jeśli dotychczas nie zostały ustawione (jego właścicielem powinien być uwierzytelniany użytkownik oraz powinny być do niego ustawione uprawnienia 700). Polecenie tworzy także plik *authorized_keys*, jeśli zachodzi taka potrzeba. To pomaga uniknąć wielu kłopotów związanych z konfigurowaniem kluczy SSH wynikających z niewłaściwych uprawnień do lokalnego bądź zdalnego katalogu *~/.ssh* albo lokalnych plików kluczy.

Po zakończeniu działania polecenia `ssh-copy-id` powinno nam się udać nawiązać połączenie przez SSH ze zdalnym serwerem bez podawania hasła. Jeśli ustawiliśmy hasło do klucza SSH, to teraz zostanie wyświetlony monit o jego podanie. Mam jednak nadzieję, że wybrałeś inne hasło do klucza niż hasło dostępu do zdalnego serwera. Dzięki temu będziesz mógł łatwiej zaobserwować, w jaki sposób działają klucze.

Wyłączanie uwierzytelniania za pomocą hasła

Gdy jesteś w stanie zalogować się do komputera przez SSH z wykorzystaniem kluczy, możesz wyłączyć możliwość uwierzytelniania za pomocą hasła. Podczas realizacji tego kroku należy oczywiście zachować ostrożność, ponieważ jeśli z jakiegoś powodu Twoje klucze przestaną działać, a wyłączysz sprawdzanie hasła, to ryzykujesz zablokowaniem dostępu do serwera. Jeśli dokonujesz tranzycji maszyny używanej przez kilka osób z uwierzytelniania za pomocą haseł na klucze, powinieneś zadbać o to, aby przed wyłączeniem uwierzytelniania za pomocą haseł wszyscy użytkownicy przesłali swoje klucze na serwer. W przeciwnym razie ktoś z uprawnieniami użytkownika *root* będzie musiał ręcznie zaktualizować plik *~/.ssh/authorized_keys* kluczem publicznym takich użytkowników.

Aby wyłączyć uwierzytelnianie za pomocą haseł, nawiąż połączenie przez SSH ze zdalnym serwerem i zdobądź uprawnienia użytkownika *root*. Następnie wyedytuj plik */etc/ssh/sshd_config* i zmień wiersz

```
PasswordAuthentication yes
```

na

```
PasswordAuthentication no
```

Następnie uruchom ponownie usługę SSH. W zależności od systemu należy w tym celu skorzystać z jednego z następujących poleceń:

```
$ sudo service ssh restart
$ sudo service sshd restart
```

Ponieważ nie chcesz dopuścić do zablokowania dostępu do serwera, zachowaj aktywną bieżącą sesję SSH. Zamiast tego otwórz nowy terminal i spróbuj połączyć się przez SSH z serwerem. Jeśli możesz zalogować się przez SSH, to znaczy, że Twoje klucze działają, a tranzycja przebiegła pomyślnie. Jeśli nie, uruchom polecenie `ssh` z opcją `-vvv`, aby uzyskać bardziej szczegółowe błędy. Dla bezpieczeństwa należy także cofnąć zmiany w pliku */etc/ssh/sshd_config* i uruchomić usługę SSH ponownie. Dzięki temu uzyskasz pewność, że nie zostaniesz całkowicie zablokowany podczas rozwiązywania problemów.

Praca z kluczami SSH chronionymi hasłem

Niektórym administratorom podoba się wygoda korzystania z kluczy SSH, które zostały stworzone bez hasła. Możesz zalogować się przez SSH natychmiast, bez konieczności wprowadzania hasła. Może to być bardzo wygodne. Jeśli korzystasz z systemu kontroli wersji, takiego jak git, przez SSH, to prawdopodobnie wolałbyś uniknąć konieczności wprowadzania hasła za każdym razem, gdy pobierasz źródła ze zdalnych repozytoriów lub wprowadzasz zmiany. Wadą tego podejścia jest to, że bez klucza SSH zabezpieczonego hasłem bezpieczeństwo Twoich serwerów jest tylko tak dobre, jak bezpieczny jest Twój prywatny klucz SSH. Jeśli ktoś zdobędzie plik *~/.ssh/id_rsa*, będzie mógł natychmiast uzyskać dostęp do wszystkich serwerów, do których ma dostęp właściciel tego pliku.

W przypadku klucza SSH zabezpieczonego hasłem, nawet jeśli dojdzie do przejęcia klucza prywatnego, napastnik nadal będzie musiał odgadnąć hasło, aby odblokować klucz. W ten sposób co najmniej zyskamy czas na stworzenie i zainstalowanie nowego klucza. W zależności od tego, kim jest napastnik, może to oznaczać, że klucz chroniony hasłem nigdy nie zostanie złamany. Stosowanie kluczy zabezpieczonych hasłem jest szczególnie ważne w przypadku, gdy klucze są przechowywane w systemach, w których konto *root* jest współdzielone z innymi administratorami. Bez zabezpieczania hasłem każdy administrator z uprawnieniami użytkownika *root* mógłby posłużyć się Twoimi poświadczeniami w celu zalogowania się na serwerach.

W przypadku stosowania kluczy SSH zabezpieczonych hasłem nie musisz rezygnować z wygody. Dostępne są narzędzia, dzięki którym posługiwanie się kluczami SSH chronionymi hasłami jest prawie tak samo wygodne, jak korzystanie z każdej innej metody, a dodatkowo gwarantuje Ci większe bezpieczeństwo. Podstawowym narzędziem, dzięki któremu zarządzanie hasłami zabezpieczającymi klucze SSH staje się łatwiejsze, jest program `ssh-add`. Program ten jest częścią narzędzia *ssh-agent*, które pozwala wprowadzić hasło raz, a następnie buforować odblokowany klucz w pamięci RAM z wykorzystaniem agenta SSH. W większości współczesnych komputerów stacjonarnych z systemem Linux agent SSH działa w tle (lub za pośrednictwem wrappera, takiego jak *keyring* w środowisku Gnome). Domyślnie klucz jest buforowany w pamięci RAM na czas nieokreślony (do chwili wyłączenia systemu). Nie polecam jednak stosowania takiej praktyki. Zamiast tego zwykle stosuję narzędzie *ssh-add* w podobny sposób do buforowania haseł `sudo`. Podaję konkretny czas, przez jaki klucz ma być przechowywany w pamięci podręcznej. Po tym czasie ponownie wyświetla się monit o wprowadzenie hasła.

Na przykład, gdybym chciał zbuforować klucz przez 15 minut, podobnie jak hasło `sudo` w niektórych systemach, mógłbym wprowadzić następujące polecenia:

```
$ ssh-add -t 15m
Enter passphrase for /home/kyle/.ssh/id_rsa:
Identity added: /home/kyle/.ssh/id_rsa (/home/kyle/.ssh/id_rsa)
Lifetime set to 900 seconds
```

Zwróćmy uwagę, że podałem czas w minutach, dzięki dodaniu przyrostka „m” na końcu wartości za argumentem `-t`; w przeciwnym razie wartość zostałaby zinterpretowana jako czas w sekundach. Zazwyczaj klucz buforuje się na nieco dłużej. Dla przykładu, aby zbuforować klucz przez godzinę, można wprowadzić następujące polecenie:

```
$ ssh-add -t 1h
Enter passphrase for /home/kyle/.ssh/id_rsa:
```

```
Identity added: /home/kyle/.ssh/id_rsa (/home/kyle/.ssh/id_rsa)
Lifetime set to 3600 seconds
```

Od tego momentu aż do chwili, gdy ważność klucza wygaśnie, możesz zalogować się przez SSH do serwerów i skorzystać z takich narzędzi jak Git bez wpisywania hasła. Gdy ten czas upłynie, to następnym razem, gdy skorzystasz z SSH, wyświetli się pytanie o hasło. Wtedy możesz zadecydować, aby uruchomić polecenie `ssh-add` ponownie. Jeśli klucz, który chcesz dodać, nie znajduje się w domyślnej lokalizacji, po prostu dodaj ścieżkę do klucza na końcu polecenia `ssh-add`:

```
$ ssh-add -t 1h ~/.ssh/workkey
```

Podoba mi się korzystanie z tego narzędzia tak, jak z osobistego timera. Kiedy rano zaczynam pracę, obliczam liczbę godzin lub minut od chwili obecnej do momentu, kiedy chcę iść na obiad, i ustawiam timer `ssh-add` na tę liczbę. Następnie pracuję tak jak zwykle, a kiedy następnym razem w efekcie polecenia `git push` lub `ssh` wyświetli się pytanie o hasło, to zdaję sobie sprawę, że właśnie nadszedł czas, aby udać się na przekąskę. Gdy wracam z obiadu, robię to samo, aby uzyskać powiadomienie o tym, kiedy będzie pora, by zakończyć pracę tego dnia.

Oczywiście, wykorzystywanie takiego narzędzia jak to oznacza, że jeśli napastnikowi uda się złamać zabezpieczenia maszyny podczas jednego z okien czasowych, podczas których klucz SSH jest zbuforowany w pamięci podręcznej, będzie on miał dostęp do wszystkiego tego, do czego ma dostęp właściciel klucza, bez konieczności wprowadzania hasła. Jeśli pracujesz w środowisku, w którym jest to zbyt duże zagrożenie, po prostu stosuj krótkie odcinki czasu `ssh-add` albo uruchamiaj polecenie `ssh-add -D`, aby usunąć wszystkie klucze zapisane w pamięci podręcznej za każdym razem, gdy pozostawiasz swoje stanowisko komputerowe bez opieki. Możesz nawet spowodować, aby polecenie `ssh-add -D` było wywoływane za każdym razem, gdy uruchamiasz polecenie `lock` lub włączasz wygaszacz ekranu, tak by pamięć podręczna kluczy była czyszczona każdorazowo, gdy chcesz zablokować komputer.

AppArmor[1]

Model uprawnień systemu Unix od dawna jest używany do blokowania dostępu użytkownikom i programom. Chociaż model ten nieźle się sprawdza, nadal istnieją obszary, gdzie dodatkowe mechanizmy kontroli dostępu mogą okazać się przydatne. Na przykład wiele usług nadal działa z tożsamością użytkownika *root*. Z tego powodu, jeśli zostaną wykorzystane, to potencjalnie napastnik może uruchamiać polecenia w pozostałej części systemu jako użytkownik *root*. Istnieje kilka sposobów radzenia sobie z tym problemem, w tym piaskownice, „więzienia” `chroot` i tak dalej. Do dystrybucji Ubuntu dołączono domyślnie instalowany system o nazwie AppArmor, który dodaje mechanizmy kontroli dostępu do specyficznych usług systemowych.

AppArmor bazuje na zasadzie najmniejszych uprawnień — czyli podejmuje próbę wprowadzenia przydzielenia programom minimalnego zestawu uprawnień, jakie są niezbędne do ich funkcjonowania. AppArmor działa za pośrednictwem zbioru reguł przypisanych do

[1] Rankin, Kyle; Hill, Benjamain Mako, *The Official Ubuntu Server Book*, wydanie trzecie, © 2014. Przedruk za zgodą Pearson Education, Inc., Nowy Jork.

konkretnych programów. Reguły te określają na przykład, które pliki lub katalogi program może czytać i zapisywać oraz które może tylko czytać. Kiedy aplikacja, która jest zarządzana przez AppArmor, narusza te mechanizmy kontroli dostępu, AppArmor zaczyna działać, zapobiega naruszeniom i rejestruje zdarzenie w logu. Niektóre usługi uwzględniają domyślnie egzekwowane profile AppArmor. W kolejnych wydaniach dystrybucji Ubuntu dodawanych jest coraz więcej takich profili. Oprócz profili domyślnych w repozytorium *universe* dostępny jest pakiet *apparmor-profile*, który możesz zainstalować, aby dodać nowe profile dla innych usług. Jeśli poznasz składnię reguł AppArmor, możesz nawet dodawać własne profile.

Prawdopodobnie najprostszym sposobem pokazania, jak działa AppArmor, jest użycie przykładowego programu. Jednym z programów automatycznie zarządzanych przez AppArmor w dystrybucji Ubuntu jest serwer DNS BIND, dlatego najpierw zainstaluję pakiet BIND za pomocą polecenia `sudo apt-get install bind9`. Po zainstalowaniu pakietu można skorzystać z programu `aa-status`, aby dowiedzieć się, czy AppArmor już nim zarządza:

```
$ sudo aa-status
apparmor module is loaded.
5 profiles are loaded.
5 profiles are in enforce mode.
/sbin/dhclient3
/usr/lib/NetworkManager/nm-dhcp-client.action
/usr/lib/connman/scripts/dhclient-script
/usr/sbin/named
/usr/sbin/tcpdump
0 profiles are in complain mode.
2 processes have profiles defined.
1 processes are in enforce mode :
/usr/sbin/named (5020)
0 processes are in complain mode.
1 processes are unconfined but have a profile defined.
/sbin/dhclient3 (607)
```

Na podstawie wyniku powyższego polecenia możemy się dowiedzieć, że załadowano profil */usr/sbin/named*, który działa w trybie `enforce`, oraz że AppArmor zarządza aktualnie uruchomionym procesem */usr/sbin/named* (PID 5020).

Profile AppArmor

Profile AppArmor są przechowywane w pliku */etc/apparmor.d/*, a ich nazwy odpowiadają binariom, którymi te profile zarządzają. Na przykład profil usługi */usr/sbin/named* jest zapisany w pliku */etc/apparmor.d/usr.sbin.named*. Wystarczy przyjrzeć się zawartości tego pliku, aby zorientować się, jak działają profile AppArmor i jakiego rodzaju ochronę zapewniają:

```
# vim:syntax=apparmor
# Last Modified: Fri Jun 1 16:43:22 2007
#include <tunables/global>
/usr/sbin/named {
    #include <abstractions/base>
    #include <abstractions/nameservice>

    capability net_bind_service,
    capability setgid,
    capability setuid,
```

```
    capability sys_chroot,

    # plik /etc/bind powinien być dostępny dla usługi bind tylko do odczytu
    #  /var/lib/bind jest potrzebny dla plików aktualizowanej dynamicznie strefy (oraz dziennika).
    # /var/cache/bind jest używany dla danych slave/stub, ponieważ nie jesteśmy ich
    #źródłem.
    # Patrz /usr/share/doc/bind9/README.Debian.gz
    /etc/bind/** r,
    /var/lib/bind/** rw,
    /var/lib/bind/ rw,
    /var/cache/bind/** rw,
    /var/cache/bind/ rw,

    # niektórzy są zwolennikami umieszczania logów w pliku /var/log/named/
    /var/log/named/** rw,

    # pakiet dnscvsutil
    /var/lib/dnscvsutil/compiled/** rw,

    /proc/net/if_inet6 r,
    /usr/sbin/named mr,
    /var/run/bind/run/named.pid w,
    # obsługa dla resolvconf
    /var/run/bind/named.options r,
}
```

Dla przykładu przyjrzyjmy się poniższemu fragmentowi z tego pliku:

```
/etc/bind/** r,
/var/lib/bind/** rw,
/var/lib/bind/ rw,
/var/cache/bind/** rw,
/var/cache/bind/ rw,
```

Składnia tych plików jest dość prosta. Najpierw podana jest ścieżka do pliku lub katalogu, a za nią dozwolone uprawnienia. Dozwolone są także symbole wieloznaczne. W związku z tym na przykład zapis `/etc/bind/**` dotyczy rekurencyjnie wszystkich plików z katalogu */etc/bind*. Pojedynczy symbol * miałby zastosowanie wyłącznie do plików w bieżącym katalogu. W przypadku tej reguły można zauważyć, że usługa */usr/sbin/named* może jedynie czytać pliki w tym katalogu, natomiast nie może ich zapisywać.

To ma sens, ponieważ ten katalog zawiera same pliki konfiguracyjne BIND — program *named* nigdy nie powinien ich nadpisywać. Drugi wiersz fragmentu zezwala programowi *named* na odczyt i zapis do plików lub folderów w katalogu */var/lib/bind/*. To też ma sens — BIND może (między innymi) zapisywać tutaj pliki stref pomocniczych, a ponieważ informacje do tych plików są zapisywane każdorazowo przy zmianie strefy, usługa *named* potrzebuje tam uprawnień zapisu.

Tryby enforce i complain

Prawdopodobnie zauważyłeś, że w wyniku działania polecenia `aa-status` są informacje na temat dwóch trybów: `enforce` i `complain`. W trybie `enforce` AppArmor aktywnie blokuje wszelkie próby naruszenia jego profilu przez programy. W trybie `complain` AppArmor po prostu rejestruje próby, ale ich nie blokuje. Programy `aa-enforce` i `aa-complain` pozwalają

zmienić profil tak, by działał w trybie `enforce` lub `complain`. Jeśli zatem mój program */usr/sbin/named* miał potrzebę zapisu do pliku w katalogu */etc/bind* lub jakimś innym katalogu, w którym nie miał zezwolenia zapisu, to albo mogłem zmodyfikować profil AppArmor, aby na to pozwolić, albo ustawić go na tryb `complain`:

```
$ sudo aa-complain /usr/sbin/named
Setting /usr/sbin/named to complain mode
```

Jeśli później zechcę ustawić tryb `enforced` ponownie, mogę skorzystać z polecenia `aa-enforce` w ten sam sposób:

```
$ sudo aa-enforce /usr/sbin/named
Setting /usr/sbin/named to enforce mode
```

Gdybym zdecydował się zmienić domyślny zestaw w pliku */etc/apparmor.d/usr.sbin.named*, musiałbym zadbać o ponowne załadowanie AppArmor, tak aby zmiany stały się widoczne. Aby to osiągnąć, możesz uruchomić skrypt inicjalizacji systemu AppArmor i przekazać do niego opcję `reload`:

```
$ sudo /etc/init.d/apparmor reload
```

Podczas modyfikowania reguł AppArmor należy zachować ostrożność. Kiedy po raz pierwszy rozpoczniesz modyfikowanie reguł, możesz ustawić określoną regułę na tryb `complain`, a następnie monitorować wszelkie nieprawidłowości w pliku */var/log/syslog*. Gdyby na przykład usługa */usr/sbin/named* była w trybie `enforce` i ujałbym w komentarz wiersz w pliku profilu */usr/sbin/named*, który gwarantował dostęp do odczytu do */etc/bind/***, a następnie ponownie załadował AppArmor i zrestartował BIND, to nie tylko BIND nie wystartowałby (ponieważ nie mógłby odczytać swoich plików konfiguracyjnych), ale także jądro wygenerowałoby czytelny wpis w dzienniku */var/log/syslog* informujący o zablokowanej próbie:

```
Jan 7 19:03:02 kickseed kernel: [ 2311.120236]
    audit(1231383782.081:3): type=1503 operation="inode_permission"
    requested_mask="::r" denied_mask="::r" name="/etc/bind/named.conf"
    pid=5225 profile="/usr/sbin/named" namespace="default"
```

Konwencje AppArmor w Ubuntu

Na poniższej liście wyszczególniono popularne katalogi i pliki używane przez AppArmor, włącznie z tymi, w których ten program przechowuje pliki konfiguracyjne i zapisuje logi:

- ***/etc/apparmor/***: ten katalog zawiera podstawowe pliki konfiguracyjne dla programu AppArmor. Należy jednak zwrócić uwagę, że nie zawiera on reguł AppArmor.
- ***/etc/apparmor.d/***: w tym katalogu są zapisane wszystkie reguły AppArmor, włącznie z podkatalogami zawierającymi różne zbiory plików nagłówkowych (ang. *include files*), do których odnoszą się określone zestawy reguł.
- ***/etc/init.d/apparmor***: w tym pliku jest zapisany skrypt inicjalizacji AppArmor. Domyślnie AppArmor jest włączony.
- ***/var/log/apparmor/***: w tym katalogu AppArmor przechowuje swoje logi.
- ***/var/log/syslog/***: kiedy reguła AppArmor zostanie naruszona w trybie `enforce` lub `complain`, wtedy jądro wygeneruje wpis w standardowym logu systemowym.

Zdalne logi

Dzienniki systemu (zwane popularnie logami) są ważnym mechanizmem rozwiązywania problemów na serwerze, ale najbardziej przydają się w sytuacji, gdy napastnikowi uda się włamać na serwer. Logi systemowe zawierają dane o wszystkich próbach logowania, zarówno lokalnych, jak i przez SSH, wszystkich próbach użycia programu `sudo`, załadowanych modułach jądra, zamontowanych dodatkowych systemach plików, a jeśli używamy programowej zapory firewall z włączonym mechanizmem logowania, to w logu można znaleźć interesujące informacje dotyczące ruchu sieciowego od intruza. W przypadku serwerów WWW, bazy danych lub aplikacji w logach pojawiają się również dodatkowe informacje dotyczące prób dostępu do tych systemów.

Problem w tym, że cyberprzestępcy wiedzą, jak przydatne są dzienniki i jak wiele informacji mogą ujawnić, dlatego każdy przeciętnie inteligentny stara się zmodyfikować wszystkie logi systemowe, które mogłyby ujawnić jego ślady. Z tego względu jednym z pierwszych działań wykonywanych przez wiele rootkitów oraz innych skryptów do przeprowadzania ataków jest usunięcie lokalnych logów i zadbanie o to, aby ich skrypty nie generowały nowych logów.

Administrator dbający o bezpieczeństwo powinien pamiętać, aby wszystkie logi systemowe, które mogą być przydatne po ataku, były dodatkowo przechowywane na oddzielnym komputerze. Scentralizowane logi są przydatne do ogólnych zadań dotyczących rozwiązywania problemów, ale również znacznie utrudniają napastnikom zacieranie śladów, ponieważ aby ukryć swoje działania, napastnik musiałby nie tylko włamać się na oryginalny serwer, ale także znaleźć sposób na włamanie na zdalny serwer logowania. W niektórych firmach obowiązują również przepisy, zgodnie z którymi niektóre kluczowe logi (np. zawierające próby logowania) muszą być przechowywane przez długi czas na oddzielnym serwerze.

Dostępnych jest wiele systemów, takich jak między innymi Splunk i Logstash, które nie tylko zbierają logi z serwerów, ale również potrafią je indeksować i udostępniają interfejs, który administratorzy mogą wykorzystać do szybkiego ich przeszukiwania. Wiele z tych usług dostarcza własnego agenta, który może być zainstalowany w systemie w celu ułatwienia zbierania logów, jednak prawie wszystkie obsługują zbieranie logów za pośrednictwem standardowego protokołu sieciowego syslog.

Zamiast opisywania w tym miejscu wszystkich dostępnych systemów zbierania logów omówię, jak należy skonfigurować klienty do wysyłania logów na zdalny serwer syslog, a w przypadku gdy nie ma centralnego serwera syslog, podam kilka prostych kroków, które pozwalają skonfigurować prosty centralny serwer syslog. Jako przykład serwera syslog wybrałem *rsyslog*. To dlatego, że obsługuje klasyczną składnię konfiguracji syslog, posiada szereg dodatkowych funkcji dla administratorów, którzy chcą dostroić konfigurację serwera, oraz jest dostępny dla wszystkich głównych dystrybucji systemu Linux.

Konfiguracja systemu Remote Syslog po stronie klienta

Skonfigurowanie klienta syslog tak, aby dostarczał logi na zdalny serwer, jest stosunkowo łatwe. Ogólnie rzecz biorąc, należy wyedytować plik konfiguracyjny syslog (w przypadku systemu rsyslog jest to plik */etc/rsyslog.conf*, w wielu przypadkach wraz z niezależnymi plikami konfiguracyjnymi w katalogu */etc/rsyslog.d*) i znaleźć ustawienia konfiguracyjne dla pliku

logu, który ma być wysyłany zdalnie. Na przykład, może mnie obowiązywać przepis, zgodnie z którym wszystkie logi uwierzytelniania mają być wysyłane do zdalnego źródła. W systemach bazujących na Debianie te logi są zapisane w pliku */var/log/auth.log*. Podczas przeglądania plików konfiguracyjnych powinienem zobaczyć wiersz, który opisuje, jaki typ zdarzeń pojawi się w tym logu:

```
auth,authpriv.* /var/log/auth.log
```

Ponieważ chcę, aby te logi były również wysyłane na zdalny serwer, muszę dodać nowy wiersz. Będzie on prawie identyczny z poprzednim, ale z tą różnicą, że ścieżka do lokalnego pliku zostanie zastąpiona lokalizacją zdalnego serwera syslog. Na przykład, jeśli zdalny serwer syslog ma nazwę *syslog1.example.com*, to albo należy dodać wiersz poniżej poprzedniego wiersza, albo utworzyć nowy plik konfiguracyjny w katalogu */etc/rsyslog.d* z następującym wierszem:

```
auth,authpriv.* @syslog1.example.com:514
```

Składnia tego wiersza jest następująca: symbol `@` dla protokołu **UDP** (ang. *User Datagram Protocol*) albo `@@` dla TCP (*Transmission Control Protocol*), nazwa hosta lub adres IP, gdzie mają być wysyłane logi, oraz opcjonalnie dwukropek i numer portu. Jeśli nie podamy portu, to wykorzystany będzie domyślny port syslog o numerze 514. Teraz, aby skorzystać z nowej konfiguracji, należy zrestartować usługę rsyslog:

```
$ sudo service rsyslog restart
```

W poprzednim przykładzie do wysyłania logów użyłem protokołu UDP. W przeszłości preferowany był protokół UDP, ponieważ podczas wysyłania logów z dużej liczby serwerów generował mniejszy ruch sieciowy. Jednak w przypadku stosowania UDP ryzykujemy utratę logów w sytuacji, gdy sieć jest przeciążona. Napastnik mógłby nawet podjąć próbę doprowadzenia do umyślnego przeciążenia sieci, aby zablokować przesyłanie logów na zdalny serwer. Chociaż stosowanie protokołu TCP jest związane z dodatkowym obciążeniem, to gwarancja dostarczenia logów warta jest tych dodatkowych kosztów. Zatem poprzedni wiersz w pliku konfiguracyjnym zmodyfikowałbym do następującej postaci:

```
auth,authpriv.* @@syslog1.example.com:514
```

Jeśli po jakimś czasie okaże się, że to generuje zbyt duże obciążenie sieci, zawsze można wrócić do UDP.

Konfiguracja systemu Remote Syslog po stronie serwera

Jeśli jeszcze nie masz jakiegoś centralnego serwera logowania w swoim systemie, stosunkowo prosto możesz go stworzyć, korzystając z systemu rsyslog. Po zainstalowaniu pakietu rsyslog należy zadbać o to, aby zdalne serwery mogły nawiązywać połączenia w tym systemie z portami 514 UDP i TCP. W związku z tym należy odpowiednio dostosować reguły zapory firewall. Następnie należy dodać następujące opcje do pliku konfiguracyjnego rsyslog (albo bezpośrednio w pliku */etc/rsyslog*, albo przez stworzenie dodatkowych plików w katalogu */etc/rsyslog.d*):

```
$ModLoad imudp
$UDPServerRun 514
$ModLoad imtcp
$InputTCPServerRun 514
```

W tej konfiguracji rsyslog będzie nasłuchiwał w porcie 514 zarówno dla protokołu UDP, jak i TCP. Należy również wprowadzić ograniczenia co do adresów IP, które mogą komunikować się z serwerem rsyslog. W związku z tym należy wprowadzić dodatkowe wiersze opisujące ograniczenia dla sieci, które mają prawo wysyłać logi do tego serwera:

```
$AllowedSender UDP, 192.168.0.0/16, 10.0.0.0/8, 54.12.12.1
$AllowedSender TCP, 192.168.0.0/16, 10.0.0.0/8, 54.12.12.1
```

Powyższe wiersze konfiguracji dają prawo przesyłania logów z sieci wewnętrznych 192.168.x.x i 10.x.x.x, a także z zewnętrznego serwera pod adresem 54.12.12.1. Oczywiście, należy zmienić adresy IP wykorzystane w powyższym przykładzie zgodnie z konfiguracją Twojej sieci.

Gdybyśmy w tej chwili zrestartowali system rsyslog, to lokalne dzienniki systemowe rozrastałyby się nie tylko w wyniku generowania zapisów logu z lokalnego hosta, ale także w wyniku przesyłania logów z systemów zdalnych. W takiej konfiguracji parsowanie i szukanie logów tylko dla konkretnego hosta jest trudne. Z tego powodu warto wprowadzić ustawienie powodujące organizowanie logów wewnątrz katalogów na podstawie nazwy hosta. W tym celu należy zdefiniować szablon dla każdego rodzaju pliku dziennika, który chcemy stworzyć. W zaprezentowanym przykładzie klienta pokazaliśmy, w jaki sposób wysyłać logi *auth.log* na zdalny serwer. Z tego powodu tutaj zaprezentujemy przykładową konfigurację, w której będziemy odbierać te logi i przechowywać lokalnie — dla każdego hosta oddzielnie — w spersonalizowanym katalogu o nazwie odpowiadającej nazwie hosta:

```
$template Rauth,"/var/log/%HOSTNAME%/auth.log"
auth.*,authpriv.* ?Rauth`
```

W pierwszym wierszu zdefiniowałem nowy szablon o nazwie `Rauth`, a następnie wskazałem, gdzie mają być przechowywane logi dla tego szablonu. Drugi wiersz przypomina wiersz konfiguracji, jaką stosowaliśmy po stronie klienta, z tą różnicą, że w tym przypadku na końcu wiersza umieściłem znak zapytania i nazwę mojego niestandardowego szablonu. Kiedy konfiguracja jest gotowa, można ponownie uruchomić rsyslog za pomocą następującego polecenia:

```
$ sudo service rsyslog restart
```

Od tej chwili powinniśmy zacząć obserwować nowe katalogi tworzone w katalogu */var/log*, odpowiadające hostom, które wysyłają logi uwierzytelniania na serwerze. Zaprezentowane powyżej wiersze szablonów można powtórzyć dla każdego typu logów, które chcemy obsługiwać. Należy jednak pamiętać, że przed skorzystaniem z szablonu należy go wcześniej zdefiniować.

Część 3. Zaawansowane techniki hartowania serwerów

W zależności od poziomu zagrożenia warto wprowadzić dla każdego z serwerów dodatkowe techniki hartowania. Do zaawansowanych technik hartowania serwerów, które opiszemy w tej części, należą szyfrowanie dysków serwera, bezpieczne alternatywy NTP oraz uwierzytelnianie dwuskładnikowe za pomocą protokołu SSH.

Szyfrowanie dysku serwera

Podobnie jak wiele bardziej zaawansowanych technik wzmacniania zabezpieczeń, szyfrowanie dysku należy do tych praktyk, które są pomijane przez wielu administratorów, jeśli nie wprowadzono obowiązku ich stosowania odpowiednimi przepisami, jeśli nie wymaga tego stopień poufności przechowywanych danych albo nie obowiązują ogólne wymogi środowiska o wysokim stopniu zabezpieczeń. W gruncie rzeczy szyfrowanie dysku wymaga dodatkowego wysiłku, obniża ogólną wydajność dysku oraz może wymagać ręcznego wpisywania hasła w celu odblokowania dysku przy starcie. Zastanawiając się nad tym, czy należy szyfrować dyski, czy nie, warto zdać sobie sprawę z tego, jakie zabezpieczenia daje ten mechanizm, a jakich nie jest w stanie zapewnić.

- Szyfrowanie chroni dane „w spoczynku". Dane są szyfrowane podczas zapisywania na dysk, ale kiedy system plików jest zamontowany, są one udostępniane w postaci niezaszyfrowanej. Gdy dysk jest odmontowany (lub gdy serwer jest wyłączony), dane są zaszyfrowane i nie mogą zostać odczytane bez znajomości hasła.
- Szyfrowanie *nie* chroni zamontowanego systemu plików. Jeśli napastnik włamie się na serwer w czasie, gdy zaszyfrowany dysk jest zamontowany (tak będzie w przypadku większości uruchomionych serwerów), będzie on w stanie odczytać dane tak samo, jakby znajdowały się one w jakimkolwiek innym niezaszyfrowanym systemie plików. Ponadto, jeśli napastnik ma uprawnienia użytkownika *root*, będzie mógł także odczytać klucz odszyfrowujący z pamięci RAM.
- Szyfrowanie jest tylko tak silne, jak użyte hasło. Jeśli do szyfrowania dysku wybierzesz słabe hasło, to napastnik w końcu je odgadnie.

Szyfrowanie dysku root

Przykłady, które przytoczę w dalszej części, dotyczą szyfrowania dysków innych niż *root*. W przypadku serwerów prostsze jest wydzielenie wrażliwych danych na zaszyfrowany dysk i pozostawienie systemowej partycji *root* w postaci niezaszyfrowanej. Dzięki temu można skonfigurować system w taki sposób, aby ładował się do wiersza poleceń i był dostępny w sieci w przypadku ponownego uruchomienia bez monitu o hasło. Niemniej jednak, jeśli środowisko jest tak wrażliwe, że nawet dysk *root* musi być zaszyfrowany, wtedy najprostszym sposobem jest konfiguracja partycji z poziomu programu instalacyjnego dystrybucji Linuksa. Można to zrobić ręcznie, w sekcji dotyczącej podziału dysku instalacyjnego, albo za pomocą automatycznego narzędzia instalacji, takiego jak *kickstart* lub *preseed*.

Szyfrowanie dysków innych niż root

Zakładam, że jeśli zdecydujesz się na zaszyfrowanie swojego dysku *root*, to prawdopodobnie podczas instalacji zaszyfrujesz także wszystkie pozostałe dyski na serwerze. Jeśli jednak nie zdecydowałeś się na zaszyfrowanie wszystkiego, to prawdopodobnie masz dysk lub partycję, których chcesz użyć do przechowywania wrażliwych informacji. W przykładach, które zaprezentuję dalej, skorzystamy z narzędzi do szyfrowania dysków *Linux Unified Key Setup* (LUKS). W szczególności użyjemy skryptu *cryptsetup*, który upraszcza proces tworzenia nowego

woluminu LUKS. Jeśli skrypt *cryptsetup* jeszcze nie został zainstalowany na serwerze, powinieneś zainstalować pakiet o tej samej nazwie właściwy dla wykorzystywanej dystrybucji.

W poniższym przykładzie skonfigurujemy zaszyfrowany wolumin na dysku */dev/dysk/sdb*, ale nie ma przeszkód, aby zamiast dysku wybrać konkretną partycję. Wszystkie polecenia wymagają uprawnień użytkownika *root*. Ostatecznie uzyskamy urządzenie dyskowe dostępne jako */dev/mapper/crypt1*. Możemy je sformatować, zamontować i traktować tak, jak każdy inny dysk.

W pierwszym kroku użyjemy narzędzia *cryptsetup* w celu stworzenia początkowego zaszyfrowanego dysku z wybranym hasłem. Następnie, zanim zaczniemy z niego korzystać, sformatujemy go z losowymi danymi:

```
$ sudo cryptsetup --verbose --verify-passphrase luksFormat /dev/sdb
WARNING!
========
This will overwrite data on /dev/sdb irrevocably.
Are you sure? (Type uppercase yes): YES
Enter passphrase:
Verify passphrase:
Command successful.
```

W tym momencie stworzyliśmy zaszyfrowany dysk LUKS na urządzeniu */dev/sdb*. Zanim jednak będzie można go używać, należy otworzyć urządzenie (co spowoduje wyświetlenie monitu o hasło) i zmapować dysk na urządzenie dostępne w katalogu */dev/mapper/*, które można zamontować:

```
$ sudo cryptsetup luksOpen /dev/sdb crypt1
Enter passphrase for /dev/sdb:
```

Składnia tego polecenia umożliwia przekazanie pakietowi *cryptsetup* polecenia `luksOpen` z argumentami w postaci urządzenia LUKS, które chcesz otworzyć, oraz etykiety, którą chcesz przypisać do tego urządzenia. Etykieta będzie nazwą, która pojawi się w katalogu */dev/mapper*, zatem w poprzednim przykładzie po wykonaniu polecenia zostanie utworzone urządzenie */dev/mapper/crypt1*.

Gdy urządzenie */dev/mapper/crypt1* istnieje, można je sformatować z określonym systemem plików i zamontować tak, jak każdy inny dysk:

```
$ sudo mkfs -t ext4 /dev/mapper/crypt1
$ sudo mount /dev/mapper/crypt1 /mnt
```

Zazwyczaj chcemy tak skonfigurować system, aby urządzenie wyświetliło się w taki sam sposób po każdym uruchomieniu. Podobnie jak plik */etc/fstab*, który jest używany do mapowania urządzeń na punkty montowania w czasie rozruchu, istnieje plik */etc/crypttab*, który można wykorzystać do zmapowania określonego urządzenia na etykietę, którą chcemy do niego przypisać. Tak jak w przypadku nowoczesnych plików */etc/fstab*, zalecane jest odwołanie się do identyfikatora UUID przypisanego do urządzenia. Identyfikator `uuid` można odczytać za pomocą narzędzia `blkid`:

```
$ sudo blkid /dev/sdb
/dev/sdb: UUID="0456899f-429f-43c7-a6e3-bb577458f92e" TYPE="crypto_LUKS"
```

Następnie należy zaktualizować plik */etc/cryptab* poprzez podanie etykiety, którą chcemy przypisać woluminowi (w naszym przykładzie *crypt1*), pełnej ścieżki do dysku, wartości `none` w polu pliku klucza oraz `lux` jako ostatniej opcji. Pełny wpis w naszym przypadku będzie wyglądać w następujący sposób:

```
$ cat /etc/crypttab
# <target name> <source device> <key file> <options>
crypt1 /dev/disk/by-uuid/0456899f-429f-43c7-a6e3-bb577458f92e none luks
```

Jeśli skonfigurowałeś plik */etc/crypttab*, w momencie rozruchu zostaniesz poproszony o hasło. Zwróćmy uwagę, że w tym przykładzie nie skonfigurowaliśmy pliku klucza. To było zrobione celowo, ponieważ plik klucza w przypadku niezaszyfrowanego systemu plików prawdopodobnie byłby dostępny dla napastnika, który uzyskałby dostęp do wyłączonego serwera. Wtedy mógłby on odszyfrować dysk.

Bezpieczne alternatywy serwera NTP

Dokładny czas jest bardzo ważny na serwerach nie tylko jako sposób synchronizacji wyjścia logów pomiędzy hostami, ale także dlatego, że większość oprogramowania obsługi klastrów bazuje na dokładnej synchronizacji zegarów. Większość hostów w celu odpytywania zdalnego serwera o dokładny czas korzysta z usługi o nazwie **NTP** (ang. *Network Time Protocol*). Do ustawienia czasu na serwerze potrzebne są uprawnienia użytkownika *root*, dlatego demon NTP (`ntpd`) zwykle działa w Twoim systemie w tle jako *root*.

Wyobrażam sobie, że większość administratorów, planując bezpieczeństwo, nie myśli o NTP. Jest to jeden z tych protokołów, które przyjmujemy za pewnik; jednak ostatecznie większość administratorów na potrzeby NTP korzysta z zewnętrznego źródła czasu (np. *nist.gov*). Ponieważ NTP korzysta z protokołu UDP, napastnik może wysłać złośliwą, fałszywą odpowiedź NTP przed legalnym serwerem. Ta odpowiedź może po prostu zawierać nieprawidłowy czas. Wysłanie go na serwer może doprowadzić do niestabilności. Co więcej, ponieważ *ntpd* działa z uprawnieniami *root*, w przypadku braku bezpiecznej walidacji odpowiedzi NTP istnieje możliwość przeprowadzenia ataku *man-in-the-middle*, w wyniku którego napastnik może uruchomić kod jako *root*.

Jedną z alternatyw dla NTP jest *tlsdate* — projekt open source, który wykorzystuje fakt, że uzgadnianie w protokole TLS zawiera informacje na temat czasu. Korzystając z *tlsdate*, można zainicjować połączenie TLS przez TCP ze zdalnego serwera, któremu ufamy, i pobrać jego czas. Znaczniki czasu w TLS nie są tak samo dokładne, jak w przypadku NTP, lecz powinny być wystarczająco dokładne do normalnego użytkowania. Ponieważ *tlsdate* korzysta z TCP i używa TLS do walidacji zdalnego serwera, przesyłanie złośliwych odpowiedzi przez napastników jest znacznie trudniejsze.

Projekt *tlsdate* jest dostępny w repozytorium *https://github.com/ioerror/tlsdate*, natomiast uniwersalne instrukcje instalacji można znaleźć pod adresem *https://github.com/ioerror/tlsdate/blob/master/INSTALL*. Ponadto *tlsdate* jest już dostępny jako pakiet dla wielu popularnych dystrybucji Linuksa. W związku z tym do wyszukania pakietu *tlsdate* najpierw należy użyć standardowego narzędzia do instalacji pakietów. Jeśli pakiet nie istnieje, to zawsze można pobrać kod źródłowy z wyżej wymienionej witryny i przeprowadzić standardowy proces kompilacji:

```
./autogen.sh
./configure
make
make install
```

Projekt *tlsdate* zawiera skrypt `systemd` lub `init` (w zależności od dystrybucji), który można uruchomić za pomocą polecenia `service tlsdated start`. Po uruchomieniu skrypt zauważy zmiany w sieci i okresowo będzie dokonywać „w tle" synchronizacji zegara. Jeśli chcesz sprawdzić `tlsdate` ręcznie, możesz ustawić zegar za pomocą następującego polecenia:

```
$ sudo tlsdate -V
Sat Jul 11 10:45:37 PDT 2015
```

Domyślnie `tlsdate` używa w roli zaufanego serwera serwisu *google.com*. Aby określić inny host, można w wierszu polecenia użyć opcji `-H`:

```
$ sudo tlsdate -V -H myserver.com
```

Aby zmienić wartości domyślne, należy wyedytować plik */etc/tlsdate/tlsdated.conf* i znaleźć sekcję `source`:

```
# Host configuration.
source
host google.com
port 443
proxy none
end
```

Zmień host na dowolny serwer, którego chciałbyś użyć. Na przykład, możesz wybrać w sieci kilka hostów, które odpytują zewnętrzne źródło czasu, i zlecić pozostałym serwerom używanie w roli źródła czasu tych wewnętrznych zaufanych hostów. Wewnętrzne zaufane serwery muszą po prostu serwować jakiś rodzaj usługi TLS (np. HTTPS).

Uwierzytelnianie dwuskładnikowe za pomocą SSH

Wyłączenie w usłudze SSH uwierzytelniania za pomocą hasła i ścisłe bazowanie na kluczach to doskonały pierwszy krok w kierunku wzmocnienia połączeń SSH, ale on również nie jest pozbawiony ryzyka. Po pierwsze, choć być może posłuchałeś moich rad i zabezpieczyłeś swoje klucze SSH hasłem, to nie jesteś w stanie zagwarantować, że każdy użytkownik w systemie zrobił to samo. Oznacza to, że jeśli napastnikowi udało się uzyskać dostęp do komputera na krótki czas, mógł skopiować klucze i użyć ich do zalogowania się do Twojego systemu. Jednym ze sposobów zabezpieczenia się przed tego rodzaju atakami jest wymaganie dla połączeń SSH uwierzytelniania dwuskładnikowego.

W przypadku uwierzytelniania dwuskładnikowego w celu zalogowania się na serwerze użytkownik musi podać zarówno klucz SSH, jak i osobny token. Najbardziej powszechne są tokeny ważne przez jakiś czas. W przeszłości ich stosowanie wymagało noszenia ze sobą drogiego urządzenia, które aktualizowało się co 30 sekund. Obecnie istnieje wiele rozwiązań dotyczących uwierzytelniania dwuskładnikowego, które działają programowo i umożliwiają korzystanie z telefonu komórkowego zamiast sprzętowego tokena.

Dla SSH istnieje kilka różnych bibliotek uwierzytelniania dwuskładnikowego bazujących na wykorzystaniu telefonów. Niektóre działają za pomocą opcji konfiguracji klienta SSH `ForceCommand`, natomiast inne z wykorzystaniem dołączanych systemowych modułów uwierzytelniania **PAM** (ang. *Pluggable Authentication Modules*). Niektóre mechanizmy mają charakter czasowy, dlatego działają nawet wówczas, gdy Twoje urządzenie jest odłączone od sieci, natomiast inne do przesyłania kodu wykorzystują wiadomości SMS lub połączenia telefoniczne. Na potrzeby przykładów zaprezentowanych w tej części rozdziału wybrałem bibliotekę Google Authenticator. Jest ku temu kilka powodów:

- Biblioteka istnieje już kilka lat i jest dostępna w formie pakietu dla wielu dystrybucji Linuksa.
- Klient Google Authenticator jest dostępny dla wielu platform telefonów komórkowych.
- Biblioteka używa modułów PAM, dlatego można ją z łatwością włączyć na poziomie systemu bez konieczności modyfikacji plików konfiguracyjnych SSH dla każdego użytkownika.
- Zapewnia użytkownikowi zapasowe kody, które można zapisać, gdyby kiedykolwiek doszło do kradzieży telefonu.

Instalacja biblioteki Google Authenticator

Biblioteka Google Authenticator jest dostępna w formie pakietu dla różnych platform. Dlatego na przykład w systemach bazujących na Debianie można ją zainstalować za pomocą następującego polecenia:

```
$ sudo apt-get install libpam-google-authenticator
```

Jeśli pakiet dla Twojej dystrybucji nie jest dostępny, przejdź pod adres *https://github.com/google/*
↪google-authenticator/ i postępuj zgodnie z instrukcjami pobierania, budowania i instalowania oprogramowania.

Konfigurowanie kont użytkowników

Zanim wprowadzisz w modułach PAM lub w konfiguracji SSH zmiany, które mogą Cię zablokować, powinieneś przynajmniej skonfigurować pakiet Google Authenticator dla swoich administratorów. Zacznij od zainstalowania aplikacji Google Authenticator na swoim smartfonie. Powinna być dostępna za pośrednictwem tych samych mechanizmów, z jakich korzystamy przy instalowaniu innych aplikacji.

Po zainstalowaniu aplikacji następnym krokiem jest utworzenie w Twoim smartfonie nowego konta Google Authenticator. Aby to zrobić, zaloguj się na swoje konto, a następnie uruchom aplikację *google-authenticator*. Wyświetli się szereg pytań. Możesz bezpiecznie odpowiedzieć twierdząco na każde z nich. Ponieważ jednak do tej pory z pewnością już uruchomiłeś mechanizm *tlsdate* na Twoim serwerze i masz dokładny czas, zalecam domyślnie trzymać się okien 90-sekundowych zamiast powiększania ich do 4 minut. Wynik działania polecenia wygląda mniej więcej tak:

```
kyle@debian:~$ google-authenticator
Do you want authentication tokens to be time-based (y/n) y
```

```
[URL for TOTP goes here]
[QR code goes here]
Your new secret key is: NONIJIZMPDJJC9VM
Your verification code is 781502
Your emergency scratch codes are:
60140990
16195496
49259747
24264864
37385449
Do you want me to update your "/home/kyle/.google_authenticator" file (y/n) y
Do you want to disallow multiple uses of the same authentication
token? This restricts you to one login about every 30s, but it increases
your chances to notice or even prevent man-in-the-middle attacks (y/n) y
By default, tokens are good for 30 seconds and in order to compensate for
possible time-skew between the client and the server, we allow an extra
token before and after the current time. If you experience problems with poor
time synchronization, you can increase the window from its default
size of 1:30min to about 4min. Do you want to do so (y/n) n
If the computer that you are logging into isn't hardened against brute-force
login attempts, you can enable rate-limiting for the authentication module.
By default, this limits attackers to no more than 3 login attempts every 30s.
Do you want to enable rate-limiting (y/n) y
```

Jeśli zainstalowałeś bibliotekę *libqrencode*, ta aplikacja nie tylko pokaże adres URL, który możesz odwiedzić, aby dodać to konto do telefonu, ale również wyświetli na konsoli kod QR (usunąłem go z poprzedniego wyjścia). Możesz zeskanować ten kod QR z telefonu lub wprowadzić tajny klucz wymieniony w wyjściu polecenia za ciągiem „Your new secret key is:”.

Awaryjne kody „zdrapki” to jednorazowe kody, z których możesz korzystać, jeśli stracisz telefon lub usuniesz jego zawartość. Zapisz je i przechowuj w bezpiecznym miejscu, osobno od telefonu.

Konfiguracja PAM i SSH

Po skonfigurowaniu na serwerze jednego lub kilku administratorów następnym krokiem w celu skorzystania z programu *Google Authenticator* jest skonfigurowanie modułu PAM i SSH. Otwórz plik konfiguracyjny SSH PAM (często w pliku */etc/pam.d/sshd*) i na początku dodaj:

```
auth required pam_google_authenticator.so
```

W moich systemach zauważyłem, że po włączeniu w pliku konfiguracyjnym mechanizmów Google Authenticator i ChallengeResponseAuthentication podczas logowania wyświetlało się pytanie o hasło także po wprowadzeniu mojego kodu uwierzytelniania dwuskładnikowego. Udało mi się wyeliminować to zachowanie poprzez ujęcie w komentarz poniższego wiersza:

```
@include common-auth
```

w pliku */etc/pam.d/sshd*. Trzeba jednak pamiętać, że w systemach, które nie bazują na Debianie, konfiguracja PAM może wyglądać trochę inaczej.

Po zaktualizowaniu pliku PAM ostatnim krokiem jest aktualizacja ustawień SSH. Otwórz plik */etc/ssh/sshd_config* i znajdź opcję `ChallengeResponseAuthentication`. Upewnij się, że jest ona ustawiona na `yes`. Jeśli tak nie jest, to w pliku *sshd_config* dodaj następujący wiersz:

```
ChallengeResponseAuthentication yes
```

Ponadto, ponieważ wcześniej wyłączyliśmy uwierzytelnianie za pomocą hasła i korzystamy z uwierzytelniania za pomocą klucza, musimy wprowadzić do pliku dodatkowy parametr. W przeciwnym razie SSH zaakceptuje klucz i nigdy nie zażąda od nas kodu uwierzytelniania dwuskładnikowego. Do pliku konfiguracyjnego dodaj również następujący wiersz:

```
AuthenticationMethods publickey,keyboard-interactive
```

Możesz teraz ponownie uruchomić SSH za pomocą jednego z następujących poleceń:

```
$ sudo service ssh restart
$ sudo service sshd restart
```

Po ponownym uruchomieniu SSH przy kolejnym logowaniu powinien wyświetlić się dodatkowy monit o wprowadzenie kodu uwierzytelniania dwuskładnikowego z aplikacji Google Authenticator:

```
$ ssh kyle@web1.example.com
Authenticated with partial success.
Verification code:
```

Od tego momentu przy każdej próbie logowania będziesz proszony o podanie swojego tokenu uwierzytelniania dwuskładnikowego.

Podsumowanie

Niezależnie od usługi działającej na serwerze możesz wykorzystać kilka podstawowych metod hartowania. W tym rozdziale skupiliśmy się głównie na czynnościach w zakresie hartowania, które mają zastosowanie do dowolnego serwera. W szczególności omówiliśmy hartowanie dostępu użytkownika *root* za pomocą polecenia `sudo` oraz znaczenie mechanizmów zdalnego logowania zdarzeń. Ponadto, biorąc pod uwagę, że obecnie praktycznie każdy serwer używa SSH do zdalnej administracji, omówiliśmy kilka technik hartowania tej usługi — począwszy od ogólnego hartowania konfiguracji serwera SSH, polegającego na wyłączeniu logowania jako *root*, a skończywszy na użyciu do uwierzytelniania kluczy SSH zamiast hasła. Na koniec omówiliśmy kilka zaawansowanych metod hartowania serwerów, w tym uwierzytelnianie dwuskładnikowe dla logowania przez SSH, szyfrowanie dysków serwera i alternatywy dla NTP.

Skorowidz

C

D

E

F

G

H

J

K

L

Ł

M

N

O

P

Q

R

S

Ś

T

U

Po pierwsze: zabezpiecz swoją sieć i zahartuj swój system!

W dzisiejszym świecie, w którym wiele codziennych aktywności odbywa się przez internet, bardzo dużo zależy od bezpieczeństwa serwerów. Kiedy zwykli ludzie tworzą społeczności, komunikują się i robią zakupy online, hakerzy niestrudzenie przeglądają sieć, poszukując słabych punktów. Atakują różne obiekty: mogą to być agencje rządowe, elektrownie i banki, ale równie dobrze celem może się stać jakakolwiek sieć komputerów. Chodzi o uzyskanie wrażliwych informacji, zbiorów danych osobowych czy wreszcie przejęcie kontroli nad systemem. Co gorsza, agresorzy odnoszą sukcesy nawet w przypadku sieci, w których wdrożono złożone i kosztowne zabezpieczenia.

Dzięki tej książce poznasz sprawdzone i niezbyt skomplikowane procedury, które pozwolą Ci na zahartowanie swoich danych. Zawarte tu treści przedstawiono w sposób bardzo praktyczny, z uwzględnieniem najnowszych osiągnięć w dziedzinie zabezpieczania systemów. Najpierw zapoznasz się z ogólnym ujęciem tematyki bezpieczeństwa systemów, w tym stacji roboczych, serwerów i sieci. Następnie dowiesz się, w jaki sposób zahartować specyficzne usługi, takie jak serwery WWW, poczta elektroniczna, systemy DNS i bazy danych. Na końcu książki znalazł się rozdział poświęcony reagowaniu na incydenty — to również jest wiedza potrzebna każdemu administratorowi.

Najciekawsze zagadnienia:

- Hartowanie stacji roboczych, w tym stacji roboczych administratorów
- Zabezpieczanie infrastruktury i ustawienie zapory sieciowej
- Zaawansowane hartowanie serwerów poczty elektronicznej
- Korzystanie z podstawowych i zaawansowanych właściwości usługi DNS
- Poruszanie się w sieci Tor

Kyle Rankin od wielu lat zajmuje się administrowaniem systemów informatycznych; to uznany ekspert w dziedzinie zabezpieczania infrastruktury, architektury, automatyzacji i rozwiązywania problemów z tym związanych. Jest nagradzanym felietonistą magazynu „Linux Journal" i przewodniczącym rady doradczej Purism. Często wygłasza referaty na konferencjach poświęconych oprogramowaniu open source i bezpieczeństwu, takich jak O'Reilly Security Conference, CactusCon, SCALE, OSCON, LinuxWorld Expo, Penguicon.

ISBN 978-83-283-4216-3

Cena: 59,00 zł

Pearson
Addison-Wesley