O'REILLY®

Cyberbezpieczeństwo w bashu

Jak za pomocą wiersza poleceń prowadzić działania zaczepne i obronne

Paul Troncone
Carl Albing

Helion

Tytuł oryginału: Cybersecurity Ops with bash: Attack, Defend, and Analyze from the Command Line

Tłumaczenie: Joanna Zatorska

ISBN: 978-83-283-8196-4

Helion S.A.
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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/cybeba.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Wprowadzenie do wiersza poleceń

Za pomocą interfejsu wiersza poleceń można nawiązać bliższą relację z *systemem operacyjnym* (ang. *operating system — OS*). System operacyjny dysponuje zdumiewającą liczbą funkcji, które udoskonalano przez kolejne dziesięciolecia. Niestety umiejętności interakcji z systemem operacyjnym poprzez wiersz poleceń szybko zanikły. Przyczynił się do tego rozwój *graficznych interfejsów użytkownika* (ang. *graphical user interface — GUI*), które zwykle ułatwiają korzystanie z systemu, ale kosztem szybkości i elastyczności. Ponadto dystansują użytkownika od funkcji systemowych.

Umiejętność skutecznego korzystania z wiersza poleceń jest niezbędna dla specjalistów ds. bezpieczeństwa oraz dla administratorów. Wiele narzędzi wykorzystywanych w tej branży, np. Metasploit, Nmap i Snort, wymaga biegłej znajomości wiersza poleceń. Podczas testów penetracyjnych często jedynym sposobem interakcji z systemem operacyjnym jest interfejs wiersza poleceń, szczególnie w początkowych etapach włamania.

Budowanie solidnych podstaw zaczniemy od omówienia wiersza poleceń i jego składników; następnie sprawdzimy, jak za pomocą tego interfejsu zwiększyć umiejętności związane z cyberbezpieczeństwem.

Definicja wiersza poleceń

W tej książce termin *wiersz poleceń* dotyczy różnych programów pozbawionych interfejsu graficznego, zainstalowanych w systemie operacyjnym. Zawierają one wbudowane programy i słowa kluczowe, a także umożliwiają uruchamianie skryptów. Wszystkie te funkcje są dostępne z poziomu powłoki, czyli interfejsu wiersza poleceń.

Aby skutecznie korzystać z wiersza poleceń, trzeba zrozumieć funkcje i opcje istniejących poleceń oraz sposób tworzenia sekwencji poleceń za pomocą języka skryptowego.

W tej książce omówiono ponad 40 poleceń, które można wykonać w systemach Linux i Windows, a także wiele różnych programów i słów kluczowych dostępnych w powłoce. Większość tych poleceń pochodzi z systemu Linux, ale jak się niebawem przekonasz, można je w różny sposób uruchomić w środowiskach z systemem Windows.

Dlaczego bash?

Do pisania skryptów wybraliśmy powłokę i język poleceń bash. Powłoka bash jest używana już od dziesięcioleci i jest dostępna w niemal każdej wersji Linuksa. Można jej nawet używać w systemie operacyjnym Windows. Dzięki temu bash jest idealną technologią wykorzystywaną w operacjach związanych z bezpieczeństwem oraz podczas tworzenia skryptów przeznaczonych na różne platformy. Wszechobecność basha jest szczególnie korzystna dla atakujących oraz testerów penetracyjnych, ponieważ w systemie docelowym zwykle nie można zainstalować dodatkowej infrastruktury ani interpreterów.

Przykłady wykonywane w wierszu poleceń

W tej książce znajduje się mnóstwo przykładów, które należy wykonać w wierszu poleceń. Polecenie jednowierszowe wygląda następująco:

```
ls -l
```

Jeśli polecenie jednowierszowe zwraca wynik, przykład wygląda następująco:

```
$ ls -l

-rw-rw-r-- 1 dave dave 15 Jun 29 13:49 hashfilea.txt
-rwxrw-r-- 1 dave dave 627 Jun 29 13:50 hashsearch.sh
```

Zwróć uwagę na znak $ w przykładzie polecenia zwracającego wynik. Początkowy znak $ nie wchodzi w skład polecenia, ale reprezentuje prosty znak zachęty w wierszu poleceń powłoki. Ułatwia odróżnienie polecenia (które trzeba wpisać) od jego wyniku w terminalu. Pusty wiersz oddzielający polecenie od wyniku w przedstawionych w książce przykładach nie będzie widoczny, gdy uruchomisz polecenie. Celem tego wiersza jest oddzielenie polecenia od jego wyniku.

Przykładowe polecenia w systemie Windows są wykonywane w powłoce Git Bash, a nie w wierszu poleceń systemu Windows, o ile nie określono inaczej.

Uruchamianie Linuksa i basha w systemie Windows

Powłoka bash i omawiane polecenia są domyślnie zainstalowane właściwie na wszystkich dystrybucjach Linuksa. Nie dotyczy to jednak środowiska Windows. Na szczęście polecenia systemu Linux oraz skrypty basha można uruchomić na wiele sposobów w systemach Windows. Omówimy cztery metody, polegające na użyciu technologii Git Bash, Cygwin, Windows Subsystem for Linux, wiersza polecenia systemu Windows oraz powłoki PowerShell.

Git Bash

Po zainstalowaniu Gita w systemie Windows można uruchomić wiele standardowych poleceń systemu Linux oraz powłokę bash. Git zawiera bowiem specjalną wersję basha przygotowaną do działania w Windowsie. Przykłady z tej książki warto wykonywać w środowisku *Git Bash*, ponieważ

jest ono popularne i można w nim uruchamiać standardowe polecenia systemu Linux oraz powłokę bash, a także wiele natywnych poleceń systemu Windows.

Git można pobrać z poświęconej mu witryny (*https://git-scm.com*). Po zainstalowaniu programu można uruchomić powłokę bash. W tym celu trzeba kliknąć pulpit lub folder prawym przyciskiem myszy i wybrać polecenie *Git Bash Here*.

Cygwin

Cygwin jest emulatorem Linuksa wyposażonym w pełną gamę funkcji i umożliwiającym instalację różnorodnych pakietów. Podobnie jak w programie Git Bash można w nim wykonywać nie tylko standardowe polecenia linuksowe, ale również wiele natywnych poleceń systemu Windows. Cygwin można pobrać z witryny poświęconej temu projektowi (*https://www.cygwin.com*).

Podsystem Windows dla systemu Linux

Po zainstalowaniu środowiska *Podsystem Windows dla systemu Linux* (ang. *Windows Subsystem for Linux — WSL*) w systemie Windows 10 otrzymasz natywne wsparcie dla Linuksa (a zarazem dla basha). Wykonaj następujące czynności, aby zainstalować WSL:

1. Kliknij pole wyszukiwania w systemie Windows 10.
2. Wyszukaj frazę **`Panel sterowania`** (**`Control Panel`**).
3. Kliknij *Programy i funkcje* (*Programs and Features*).
4. Kliknij *Włącz lub wyłącz funkcje systemu Windows* (*Turn Windows features on or off*).
5. Zaznacz pole *Podsystem Windows dla systemu Linux* (*Windows Subsystem for Linux*).
6. Zrestartuj system.
7. Otwórz *Sklep Windows* (*Windows Store*).
8. Poszukaj systemu Ubuntu i zainstaluj go.
9. Po zainstalowaniu systemu Ubuntu otwórz program *Wiersz polecenia systemu Windows* (*Windows Command Prompt*) i wpisz **`ubuntu`**.

Warto zauważyć, że podczas korzystania z dystrybucji Linuksa w środowisku WSL można uruchamiać skrypty w bashu i dołączyć system plików Windows, ale nie można wykonywać natywnych systemowych poleceń Windowsa, co jest możliwe w środowiskach Git Bash i Cygwin.

Po zainstalowaniu środowiska WSL można zainstalować inną wersję Linuksa, nie tylko Ubuntu, np. Kali. Wystarczy wybrać odpowiednią dystrybucję w sklepie Windows.

Wiersz polecenia systemu Windows i PowerShell

Po zainstalowaniu środowiska WSL polecenia Linuksa oraz skrypty basha można również uruchamiać bezpośrednio w *wierszu polecenia systemu Windows* oraz w powłoce *PowerShell*. Wystarczy skorzystać z polecenia `bash -c`.

W wierszu polecenia systemu Windows można np. wykonać polecenie Linuksa `pwd` względem bieżącego katalogu roboczego:

```
C:\Users\Paul\Desktop>bash -c "pwd"
/mnt/c/Users/Paul/Desktop
```

Jeśli w ramach środowiska WSL zainstalowanych jest kilka dystrybucji Linuksa, zamiast nazwy `bash` w poleceniu można użyć nazwy dystrybucji:

```
C:\Users\Paul\Desktop>ubuntu -c "pwd"
/mnt/c/Users/Paul/Desktop
```

Za pomocą tej metody można też używać pakietów zainstalowanych w dystrybucji Linuksa w środowisku WSL, o ile udostępniają interfejs wiersza poleceń, jak np. Nmap.

Ten, wydawałoby się, niewielki dodatek daje nam możliwość wykorzystania całego dostępnego w Linuksie arsenału poleceń, pakietów i basha z poziomu wiersza polecenia systemu Windows, a także ze skryptów typu Batch i skryptów powłoki PowerShell.

Podstawy wiersza poleceń

Wiersz poleceń jest ogólnym określeniem technik umożliwiających wykonywanie poleceń w interaktywnych systemach komputerowych przed wynalezieniem graficznych interfejsów użytkownika. W systemach linuksowych termin ten dotyczy interakcji z powłoką bash (lub innymi). Jedną z podstawowych funkcji basha jest uruchamianie poleceń, czyli innych programów. Po wpisaniu kilku słów w wierszu poleceń bash zakłada, że pierwsze z nich jest nazwą programu, który trzeba uruchomić, a pozostałe są jego argumentami. Aby uruchomić w bashu np. polecenie o nazwie `mkdir` i przekazać do niego dwa argumenty `-p` i `/tmp/scratch/garble`, należy wpisać następujące wyrażenie:

```
mkdir -p /tmp/scratch/garble
```

Zgodnie z powszechną konwencją opcje programów zwykle znajdują się na początku i są poprzedzone znakiem `-`, jak w przypadku opcji `-p` z poprzedniego przykładu. Polecenie z przykładu tworzy katalog o nazwie */tmp/scratch/garble*. Opcja `-p` oznacza, że użytkownik wybrał określone zachowanie polecenia, a mianowicie brak wyświetlania błędów i utworzenie (lub próbę utworzenia) wszystkich potrzebnych katalogów (np. jeśli istnieje tylko katalog */tmp*, wówczas polecenie `mkdir` utworzy najpierw katalog */tmp/scratch*, a następnie katalog */tmp/scratch/garble*).

Polecenia, argumenty, polecenia wbudowane i słowa kluczowe

Do poleceń, które można uruchomić, należą pliki, polecenia wbudowane i słowa kluczowe.

Pliki są wykonywalnymi programami. Mogą powstać w wyniku kompilacji i zawierać instrukcje maszynowe. Przykładem jest program `ls`. Ten plik znajduje się w większości systemów plików Linuksa w lokalizacji */bin/ls*.

Innym typem pliku jest *skrypt*, czyli czytelny dla ludzi plik tekstowy, napisany w jednym z kilku języków obsługiwanych przez system za pośrednictwem interpretera (programu) tego języka. Do przykładowych języków skryptowych należą bash, Python i Perl. W kolejnych rozdziałach napiszesz wiele skryptów (w bashu).

Polecenia wbudowane wchodzą w skład powłoki. Wyglądają jak pliki wykonywalne, ale w systemie plików nie istnieje żaden plik, który trzeba wczytać i wykonać. Ich działanie odbywa się w ramach powłoki. Przykładem polecenia wbudowanego jest `pwd`. Użycie polecenia wbudowanego jest szybsze i wydajniejsze. Również użytkownicy mogą definiować funkcje w powłoce, które można wykonywać podobnie jak polecenia wbudowane.

Istnieją inne słowa, które wyglądają jak polecenia, ale w rzeczywistości są częścią języka powłoki. Przykładem jest słowo `if`. Zwykle jest używane jako pierwsze słowo w wierszu poleceń, ale nie jest plikiem; jest to *słowo kluczowe*. Związana jest z nim składnia, która może być bardziej złożona niż typowy wzorzec *`polecenie -opcje argumenty`*. Słowa kluczowe omówimy pokrótce w następnym rozdziale.

Za pomocą polecenia `type` można sprawdzić, czy słowo jest słowem kluczowym, poleceniem wbudowanym, zwykłym poleceniem, czy innym elementem. Dzięki opcji `-t` uzyskasz jednowyrazowy wynik:

```
$ type -t if
keyword
$ type -t pwd
builtin
$ type -t ls
file
```

Za pomocą polecenia `compgen` można ustalić dostępne polecenia, polecenia wbudowane i słowa kluczowe. Za pomocą opcji `-c` uzyskasz listę poleceń, opcji `-b` — listę poleceń wbudowanych, a opcji `-k` — słów kluczowych:

```
$ compgen -k

if
then
else
elif
.
.
.
```

Jeśli taki podział jest na razie mylący, nie przejmuj się. Zwykle nie trzeba wiedzieć, jakiego rodzaju jest polecenie, ale warto pamiętać, że użycie poleceń wbudowanych i słów kluczowych jest znacznie bardziej wydajne niż użycie zwykłych poleceń (programów wykonywalnych w postaci zewnętrznych plików), szczególnie w przypadku wywoływania poleceń w pętli.

Standardowy strumień wejścia, wyjścia i błędów

Zgodnie z terminologią stosowaną w opisie systemów operacyjnych uruchomiony program jest *procesem*. Każdy proces w środowisku Unix/Linux/POSIX (a zatem również w systemie Windows) ma trzy różne deskryptory wejścia i wyjścia. Są to *standardowe strumienie wejścia* (*stdin*), *wyjścia* (*stdout*) i *błędów* (*stderr*).

Jak sama nazwa wskazuje, stdin jest domyślnym źródłem danych wejściowych programu — domyślnie są to znaki wpisywane na klawiaturze. Jeśli skrypt pobiera dane ze strumienia stdin, odczytuje znaki wpisywane na klawiaturze. To zachowanie można zmienić (o czym się niebawem przekonasz), aby dane były wczytywane z pliku. Strumień stdout jest domyślnym miejscem, do którego program wysyła swoje wyniki. Domyślnie wyniki są wyświetlane w oknie, w którym uruchomione są powłoka lub skrypt powłoki. Wynik działania programu można też przesłać do standardowego strumienia błędów, jednak do tego strumienia powinno się wysyłać komunikaty błędów. To programista decyduje, czy wynik zostanie przekierowany do strumienia stdout, czy stderr. Podczas pisania skryptów trzeba zatem pamiętać, aby komunikaty błędów wysyłać do strumienia stderr, a nie stdout.

Przekierowania i potoki

Jedną z większych innowacji wprowadzonych w powłoce było wdrożenie mechanizmu, dzięki któremu można przekierować miejsce pobierania danych i (lub) zmienić miejsce wysyłania wyników *bez modyfikacji samego programu*. Na przykład jeśli program `handywork` pobiera dane wejściowe ze strumienia stdin i wysyła wyniki do strumienia stdout, jego zachowanie można zmienić w następujący prosty sposób:

```
handywork < data.in > results.out
```

To polecenie uruchomi program `handywork`, jednak w tym przypadku dane wejściowe zostaną pobrane nie z klawiatury, ale z pliku o nazwie *data.in* (zakładając, że taki plik istnieje i ma odpowiedni format). Natomiast dane wyjściowe nie zostaną wyświetlone na ekranie, ale przekierowane do pliku *results.out* (który zostanie w razie potrzeby utworzony, a jeśli już istnieje, jego zawartość zostanie nadpisana). Tę technikę nazywa się *przekierowaniem*, ponieważ polega na przekierowaniu danych wejściowych z innego źródła oraz na przekierowaniu danych wyjściowych do miejsca innego niż ekran.

Jak można przekierować strumień stderr? Składnia jest podobna. Podczas przekierowania danych wyjściowych programu trzeba odróżnić od siebie strumienie stdout i stderr. W tym celu trzeba skorzystać z numerów deskryptorów plików. Stdin jest deskryptorem plików o numerze 0, stdout jest deskryptorem plików o numerze 1, a stderr jest deskryptorem plików o numerze 2, a zatem komunikaty błędów można przekierować następująco:

```
handywork 2> err.msgs
```

W ten sposób przekierujemy tylko strumień stderr, dzięki czemu wszystkie komunikaty błędów zostaną zapisane w pliku *err.msgs*.

Oczywiście wszystkie strumienie można przekierować w jednym wierszu:

```
handywork < data.in > results.out 2> err.msgs
```

Czasem potrzebne są komunikaty błędów wraz z wynikiem działania programu (jak w domyślnym mechanizmie polegającym na wyświetleniu obydwu strumieni na ekranie). W tym przypadku można skorzystać z następującej składni:

```
handywork < data.in > results.out 2>&1
```

To polecenie przesyła strumień stderr (2) w to samo miejsce co deskryptor pliku 1 (`&1`). Zauważ, że bez ampersandu komunikaty błędu zostałyby wysłane do pliku o nazwie *1*. Tego typu połączenie strumieni stdout i stderr jest tak popularne, że powstał przydatny skrót:

```
handywork < data.in &> results.out
```

Jeśli chcesz pominąć standardowy strumień wyjścia, możesz przekierować wyniki do specjalnego pliku o nazwie */dev/null*:

```
handywork < data.in > /dev/null
```

Aby wyświetlić dane wyjściowe na ekranie i jednocześnie przekierować je do pliku, skorzystaj z polecenia `tee`. Następujące polecenie wyświetli wynik programu `handywork` na ekranie i zapisze dane wyjściowe w pliku *results.out*:

```
handywork < data.in | tee results.out
```

Dzięki opcji `-a` polecenie `tee` dołączy dane wyjściowe do pliku, a nie nadpisze jego zawartości. Znak `|` reprezentuje tzw. *potok*. Umożliwia on pobieranie danych wyjściowych z jednego polecenia lub skryptu i użycia ich jako danych wejściowych innego polecenia. W tym przykładzie dane wyjściowe polecenia `handywork` są przekierowywane do polecenia `tee` w celu dalszego przetworzenia.

Jeśli wynik polecenia zostanie przekierowany za pomocą jednego znaku większości, plik zostanie utworzony lub nadpisany (jego dotychczasowa zawartość zostanie usunięta). Jeśli chcesz zachować istniejącą zawartość pliku, możesz *dołączyć* wynik do pliku za pomocą podwójnego znaku większości, jak w następującym przykładzie:

```
handywork < data.in >> results.out
```

To polecenie uruchamia program `handywork` w taki sposób, że dane ze strumienia stdout zostaną dołączone do pliku *results.out* i nie nadpiszą jego dotychczasowej zawartości.

Podobnie działa następujące polecenie:

```
handywork < data.in &>> results.out
```

Uruchamia ono program `handywork` i dołącza dane ze strumieni stdout i stderr do pliku *results.out*, zapobiegając nadpisaniu jego istniejącej zawartości.

Uruchamianie poleceń w tle

W tej książce nie będziemy korzystać tylko z poleceń jednowierszowych, ale będziemy również pisać złożone skrypty. Wykonanie niektórych z nich może być dość czasochłonne. Niekiedy czas ich działania może być tak długi, że nie warto czekać na zakończenie. W tym przypadku polecenie lub skrypt można uruchomić w tle za pomocą operatora `&`. Skrypt będzie nadal działać, a w powłoce można będzie wykonywać inne polecenia lub skrypty. Aby uruchomić w tle np. program `ping` i przekierować standardowy strumień wyjścia do pliku, można wykonać następujące polecenie:

```
ping 192.168.10.56 > ping.log &
```

W przypadku zadań wykonywanych w tle najczęściej przekierowuje się do pliku standardowe strumienie wyjścia i błędów, aby uniknąć wyświetlania na ekranie wygenerowanych przez nie danych, co może zakłócać pracę w wierszu poleceń:

```
ping 192.168.10.56 &> ping.log &
```

Uważaj, aby nie pomylić operatora & (służącego do wykonywania zadań w tle) z operatorem &> (służącym do przekierowania zarówno standardowego strumienia wyjścia, jak i standardowego strumienia błędów).

Za pomocą polecenia `jobs` można wyświetlić listę wszystkich zadań uruchomionych w tle:

```
$ jobs
[1]+ Running     ping 192.168.10.56 > ping.log &
```

Za pomocą polecenia `fg` i numeru odpowiedniego zadania można przenieść to zadanie na pierwszy plan:

```
$ fg 1
ping 192.168.10.56 > ping.log
```

Jeśli zadanie działa na pierwszym planie, można je zawiesić, naciskając kombinację klawiszy *Ctrl+Z*, a następnie poleceniem `bg` wznowić jego działanie w tle. Następnie można skorzystać z opisanych już poleceń `jobs` i `fg`.

Od wiersza poleceń do skryptu

Skrypt powłoki jest plikiem zawierającym takie same polecenia, jakie można wpisać w wierszu poleceń. Umieść co najmniej jedno polecenie w pliku, a uzyskasz skrypt powłoki. Jeśli nazwiesz ten plik *mojskrypt*, będziesz mógł go uruchomić, wpisując polecenie `bash mojskrypt`. Możesz też nadać plikowi *uprawnienia do wykonywania* (np. `chmod 755 mojskrypt`), dzięki czemu możesz go wywołać bezpośrednio: `./mojskrypt`. Zwykle w pierwszym wierszu skryptu wpisuje się następujący kod, który informuje system operacyjny o wykorzystanym języku skryptowym:

```
#!/bin/bash -
```

Oczywiście w tym przypadku zakładamy, że program bash znajduje się w katalogu */bin*. Jeśli skrypt musi być bardziej uniwersalny, można użyć następującego kodu:

```
#!/usr/bin/env bash
```

W tym przykładzie polecenie `env` poszuka lokalizacji basha. Jest to standardowy sposób radzenia sobie z problemem przenośności. Zakłada on jednak, że program `env` znajduje się w katalogu */usr/bin*.

Podsumowanie

Wiersz poleceń można porównać z fizycznym narzędziem wielofunkcyjnym. Jeśli chcesz wkręcić śrubę w kawałek drewna, powinieneś skorzystać z wyspecjalizowanego narzędzia, takiego jak śrubokręt lub wkrętarka. Jeśli jednak znajdujesz się sam w lesie i masz ograniczone zasoby, najlepiej sprawdzi się narzędzie wielofunkcyjne. Za jego pomocą możesz wkręcić śrubę w kawałek drewna, przeciąć linę, a nawet otworzyć butelkę. To samo dotyczy wiersza poleceń. Jego wartość nie wynika z perfekcji wykonania jednego określonego zadania, ale z jego wszechstronności i dostępności.

Powłoka bash i polecenia Linuksa zyskały ostatnio na popularności. W systemie Windows możesz z nich z łatwością skorzystać w środowisku Git Bash lub Cygwin. Aby uzyskać jeszcze większe możliwości, możesz zainstalować środowisko Podsystem Windows dla systemu Linux, w którym uruchomisz pełne wersje systemów Linux. Będziesz też mieć do nich dostęp w wierszu polecenia systemu Windows i w powłoce PowerShell.

W następnym rozdziale omawiamy możliwości skryptów, które mogą powtarzać uruchamianie programów, podejmować decyzje i przetwarzać w pętli dane wejściowe różnego rodzaju.

Ćwiczenia

1. Napisz polecenie, które wykona program `ifconfig` i przekieruje standardowy strumień wyjścia do pliku *ipaddress.txt*.
2. Napisz polecenie, które wykona program `ifconfig` i dołączy dane ze standardowego strumienia wyjścia do pliku *ipaddress.txt*.
3. Napisz polecenie, które skopiuje wszystkie pliki z katalogu */etc/a* do katalogu */etc/b* i przekieruje standardowy strumień błędów do pliku *copyerror.log*.
4. Napisz polecenie, które pobierze (`ls`) zawartość głównego katalogu i prześle wyniki do polecenia `more`.
5. Napisz polecenie, które wykona program *mojezadanie.sh* w tle.
6. Na podstawie następującej listy zadań napisz polecenie, które przeniesie na pierwszy plan program `ping` odpytujący witrynę Amazona:

```
[1]  Running    ping www.google.com > /dev/null &
[2]- Running   ping www.amazon.com > /dev/null &
[3]+ Running   ping www.oreilly.com > /dev/null &
```

Dodatkowe zasoby dotyczące tych ćwiczeń, a także ich rozwiązania znajdziesz w witrynie poświęconej książce (*https://www.rapidcyberops.com/*).

Grafy: przełomowa koncepcja w analizie danych!

Zapewnienie bezpieczeństwa IT przypomina wyścig zbrojeń. Trzeba bezustannie wykrywać zagrożenia i reagować na incydenty bezpieczeństwa, gdyż przeciwnicy wciąż się doskonalą i opracowują nowe metody ataku. Podobnie jak podczas wojny, należy stawiać na szybkość, zwinność, wykorzystywanie okazji, a także precyzję ataku i kontrataku. Okazuje się, że jeśli konieczny jest taki sposób działania, jedyna możliwa opcja to użycie standardowego wiersza poleceń — żaden inny program nie dorówna zwykłemu CLI zwinnością, elastycznością i dostępnością.

Ta książka jest praktycznym podręcznikiem dla inżynierów zajmujących się bezpieczeństwem. Znajdziesz w niej wyczerpujące omówienie technik stosowania CLI i powłoki bash do zbierania i analizy danych, do wykrywania włamań, w inżynierii wstecznej i do wykonywania zadań administracyjnych. Dowiesz się, jak prosto rozpocząć analizę dzienników i monitorowanie sieci. Wzbogacisz też swój arsenał pentestera o umiejętność używania funkcji wbudowanych w niemal każdą wersję Linuksa, jednak techniki te można łatwo przenosić również na systemy uniksowe, Windows czy macOS. Zawarta tu wiedza pomoże Ci wyjść obronną ręką z każdej sytuacji, gdy dla zażegnania kryzysu bezpieczeństwa trzeba będzie się wykazać zdolnością błyskawicznej reakcji i dokładnością działań.

W książce:

- wprowadzenie do wiersza poleceń i basha
- zasady defensywy i ofensywy w cyberbezpieczeństwie
- analiza danych i szkodliwego oprogramowania oraz monitorowanie dzienników
- testy penetracyjne
- tajniki pracy administratora bezpieczeństwa

Paul Troncone od ponad 15 lat zajmuje się cyberbezpieczeństwem i programowaniem. Posiada certyfikat CISSP (*Certified Information Systems Security Professional*). Pracował jako analityk ds. bezpieczeństwa, programista, pentester i wykładowca uniwersytecki.

Dr Carl Albing jest nauczycielem, naukowcem i programistą z ogromnym doświadczeniem. Pracował jako programista w dużych i małych firmach z różnych branż. Obecnie jest wykładowcą na wyższej uczelni i naukowcem.

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-8196-4
9 788328 381964
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 69,00 zł