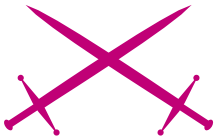



# Restrykcyjne powłoki – jak je obejść?

Dawid Gołuński

stopień trudności

**Administratorzy, chcąc ograniczyć swobodę użytkowników systemu, decydują się na ustawienie restrykcyjnej powłoki, tak aby umożliwić wykonywanie tylko wybranych komend. Niestety, utworzenie w pełni restrykcyjnego środowiska nie jest rzeczą łatwą, a najdrobniejsze przeoczenie może zezwolić sprytnemu użytkownikowi na wydostanie się z pułapki.**

Restrykcyjna powłoka (*restricted shell*) to powłoka systemowa zmodyfikowana w taki sposób, aby ograniczyć zakres możliwości użytkownika. Taka powłoka zwykle znajduje zastosowanie w momencie, gdy administrator stoi przed koniecznością utworzenia konta dla użytkownika, który nie jest w pełni zaufany. Chodzi tu o to, by zapobiec takim sytuacjom jak np. zbieranie informacji o systemie i jego użytkownikach poprzez odczyt plików konfiguracyjnych czy uruchamianie programów systemowych zdradzających istotne informacje. Czasem zachodzi też potrzeba ochrony użytkownika przed samym sobą – niewprawny użytkownik, korzystając z narzędzia, które nie jest mu do końca znane, mógłby wykasować część plików zgromadzonych na jego koncie czy też – poprzez nadanie złych uprawnień – udostępnić plik osobom niepowołanym. Przykładem zastosowania restrykcyjnej powłoki może być serwer pocztowy, na którym nie działa serwer WWW. Nie ma zatem panelu pocztowego, na którym użytkownicy mogliby sprawdzać swoją pocztę zdalnie (poprzez przeglądarkę WWW). Administrator może pokusić się tutaj o przyznanie użytkownikom restrykcyjnej powłoki, tak aby po zdalnym połączeniu możliwe było uruchomienie klienta pocztowego oraz programu *passwd* - w celu zmiany hasła do konta.

## Tworzenie restrykcyjnego środowiska

Administratorzy w celu utworzenia restrykcyjnego środowiska zazwyczaj wykorzystują standardowe powłoki dostępne w systemie, takie

### Z artykułu dowiesz się

- Czym są, oraz jak działają restrykcyjne powłoki,
- w jaki sposób zbudowane jest typowe restrykcyjne środowisko,
- poznasz sposoby na wydostanie się z ograniczonego shella,
- czego użyć do zbudowania bardziej niezawodnego restrykcyjnego środowiska.

### Powinieneś wiedzieć

- Posiadać wiedzę na temat pracy z systemową powłoką,
- znać przynajmniej podstawy działania systemów z rodziny linux/unix,
- znać podstawy języka C.

## Błąd w programie bash

Na niektórych systemach utworzenie linku *rbash* oraz ustawienie go jako shella nie przynosi zamierzonego rezultatu – powłoka nie rozpoczyna pracy w trybie restrykcyjnym. Jest to błąd, który został załatany w wersji 3.0 programu *bash*. Można ominąć ten problem poprzez utworzenie pośredniego skryptu:

```
#!/bin/bash
/bin/rbash -l
```

Taki skrypt należy zapisać jako np. */bin/rshell*, nadać mu prawo wykonywania, a następnie ustawić go jako powłokę. Po zalogowaniu powinniśmy otrzymać *rbasha* działającego już z restrykcjami. Lepiej jednak rozważyć aktualizację do wersji 3.0.

jak *bash*, *sh* czy *ksh*. Powłoki te, odpowiednio wywołane, mogą stać się powłokami restrykcyjnymi. Zwykle wystarczy wywołać plik powłoki pod nazwą zaczynającą się od litery *r*. Dla powłoki *bash*, będzie to *rbash*, dla *sh* - *rsh* itd. Wygodnie jest tutaj posłużyć się dowiązaniem symbolicznym:

```
ln -s /bin/bash /bin/rbash
```

Innym sposobem jest wywołanie powłoki z parametrem `-r`, lub `-restricted`.

Powłoka uruchomiona w takim trybie będzie nakładała ograniczenia. Najbardziej istotne z nich to:

- brak możliwości zmiany katalogu poleceniem `cd`,
- użytkownik może uruchamiać jedynie komendy zawarte w zmiennej `PATH` lub wbudowane w samą powłokę,
- brak możliwości zmiany zmiennych środowiskowych `SHELL`, `PATH`, `ENV`, `BASH _ ENV`,
- zabronione uruchamianie komend zawierających ścieżkę do pliku (np. `/bin/ls`),
- zabronione przekierowanie wyjścia aplikacji przy użyciu operatorów jak `>` czy `>>`,
- wyłączona komenda `exec`.

Po utworzeniu restrykcyjnej powłoki, w postaci symbolicznego linku – *rbash*, jesteśmy gotowi, aby przypisać ją danemu użytkownikowi. Można tego dokonać, edytując bezpośrednio plik *passwd* lub wydając komendę:

```
usermod -s /bin/rbash john
```

Niekiedy konieczne może okazać się dodanie linijki */bin/rbash* do pliku */etc/shells*, w przeciwnym razie część usług (jak np. serwer FTP) może odmówić zalogowania użytkownika.

Ustawienie samej powłoki to nie wszystko; kolejnym krokiem jest przygotowanie katalogu użytkownika. Katalog ten powinien mieć ustawione prawa 511 (najlepiej gdyby jego właścicielem był *root*, a nie użytkownik). Wewnątrz katalogu należy utworzyć plik *.bash_profile* ustawiając mu prawa dostępu 444 oraz właściciela *root*. Poza katalogiem domowym użytkownika należy utworzyć katalog, w którym umieszczone zostaną wybrane programy. Może to być np. */usr/local/rbin*. W katalogu tym umieścimy dowiązania symboliczne do aplikacji, które chcemy udostępnić, np.

```
ln -s /usr/bin/passwd \
      /usr/local/rbin/passwd
```

Ostatnim krokiem jest edycja plików konfiguracyjnych powłoki. W pliku */etc/profile* znajdują się globalne ustawienia. Aby wpisy z tego pliku nie miały wpływu na ustawienia powłoki *rbash*, jego zawartość należy zamieścić w ramach instrukcji warunkowej:

```
if [ "$0" = "-bash" ]; then
#tutaj wstawić dotychczasową zawartość#
#pliku /etc/profile                   #
fi
```

Natomiast lokalny plik ustawień *.bash_profile* powinien zawierać tylko jedną linijkę – określającą ścieżkę do katalogu z przygotowanymi wcześniej programami:

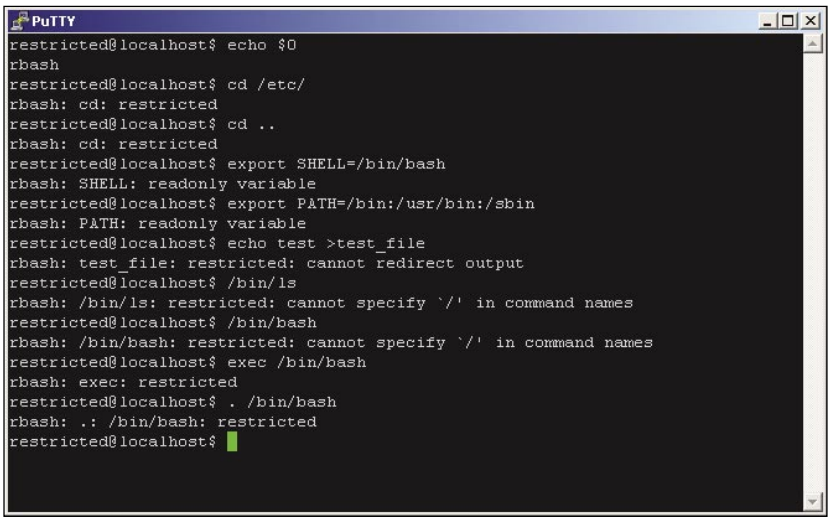
```
export PATH=/usr/local/rbin
```

Ewentualnie, jeśli chcemy odebrać użytkownikowi możliwość wykonywania niektórych poleceń wewnętrznych, musimy na końcu pliku dopisać linijkę zawierającą komendę `enable -n`, po której podajemy komendy, jakie chcemy wyłączyć. Np.:

```
enable -n pwd unset
```

## Jak to działa?

Gdy użytkownik zaloguje się do systemu (*ssh*), zostanie przeniesiony do katalogu domowego, a następnie uruchomiana jest powłoka */bin/rbash*. Powłoka wczytuje pliki konfiguracyjne poczynając od */etc/profile*. Instrukcja warunkowa zawarta na początku sprawdza, czy zerowy argument (`argv[0]`) jest równy ciągowi znaków *-bash*. W przypadku powłoki *rbash* argument `0` jest równy *-rbash* toteż polecenia zgromadzone wewnątrz warunku `if` nie zosta-




**Rysunek 1.** *Ograniczenia wprowadzane przez powłokę rbash*

ną wykonane. Powłoka przejdzie do wykonania lokalnego pliku konfiguracyjnego – *.bash_profile*, który ustawi zmienną środowiskową `PATH`. Dziwić może fakt zmiany tej zmiennej, gdyż zgodnie z tym, co wcześniej napisałem – jednym z ograniczeń powłoki *rbash* jest ustawienie zmiennej `PATH` w stan tylko do odczytu. Powodowane jest to tym, że wszystkie wymienione ograniczenia wprowadzane są dopiero po załadowaniu plików startowych (dlatego tak istotne jest ustawienie właściwych uprawnień).

## Wydostajemy się z pułapki

Utworzone według przedstawionego schematu restrykcyjne środowisko zdaje się być bezpieczne. Niestety, bezpieczeństwo to jest ściśle związane z udostępnionymi programami, jak też z – pozornie niezwiązanymi z tym środowiskiem – usługami działającymi na serwerze. Poniżej postaram się pokazać kilka przykładowych sytuacji, w których możliwe jest ominięcie zabezpieczeń powłoki.

### Operacje na plikach

Niektórzy administratorzy zapominają o fakcie, że restrykcyjna powłoka *rbash* nie zamyka użytkownika w specjalnie odseparowanym drzewie katalogów (wykorzystując funkcję `chroot()`) – blokuje ona jednie zmianę katalogu za pomocą komendy `cd`. Zapominają również o tym, że odwołania do plików zawierających ścieżkę dostępową blokowane są tylko w przypadku, gdy użytkownik próbuje odwołać się bezpośrednio do aplikacji np. */bin/ls* lub też wykonać instrukcje `exec /etc/profile` czy `source /etc/profile`. Oznacza to, że jeśli użytkownik ma dostęp do programu *pico*, to nic nie stoi na przeszkodzie, aby po wydaniu komendy: `pico /etc/passwd` ujrzał on plik *passwd*.

### Shell escape

Większość programów nie była niestety pisana pod kątem współpracy z restrykcyjnymi powłokami. Niektóre z nich zawierają wbudowaną funkcję pozwalającą na odwołanie się do powłoki (*shell escape*). Zazwyczaj funkcję tą wywołuje się przy użyciu znaku *!*. Przydaje się to np. w momencie, gdy pracując pod kontrolą klienta FTP chcemy obejrzeć listę plików w bieżącym katalogu. Nie musimy wtedy otwierać nowej sesji SSH, wystarczy bowiem, że wydamy w linii poleceń komendę: `! ls -l`. Przykłady standardowych programów dostępnych w systemie i oferujących *shell escape*: *telnet*, *mail*, *ftp*, *less*, *more*, *vi*, *gdb*, *lynx*.

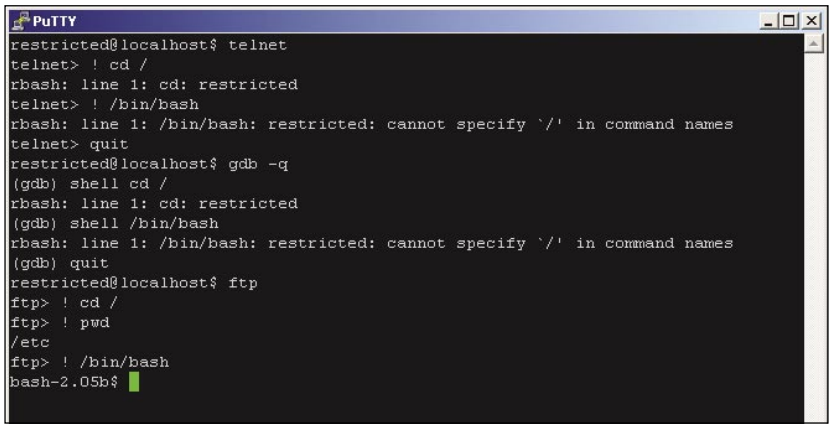
Można by się spodziewać, że przekazane komendy zostaną wykonane w oparciu o powłoką zdefiniowaną w zmiennej `SHELL` – w naszym przypadku */bin/rbash*. Niestety nie zawsze tak jest. Część programów nie zwraca uwagi na zmienne środowiskowe – wykonując przekazane polecenia przy użyciu trwale ustalonej (w kodzie programu) powłoki – */bin/bash* czy też */bin/sh*.

Ciekawym przykładem może być tutaj program *vim*. Zwraca on wprawdzie uwagę na zmienną środowiskową `SHELL` – po rozpoczęciu pracy kopiuje jej zawartość do wewnętrznej zmiennej *shell*. Problem polega jednak na tym, że ustawienia programu *vim* możemy dowolnie edytować w czasie jego działania. Wystarczą do tego dwie komendy wydane w linii poleceń, (do której dostaniemy się sekwencją klawiszy *[ESC][:]*) :

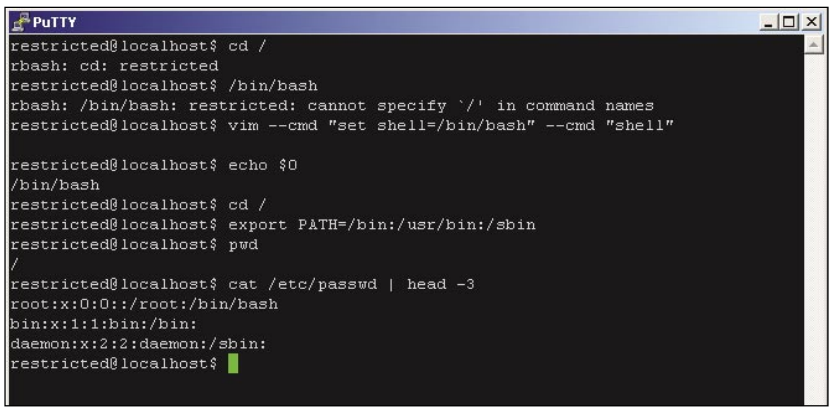
- `set shell=/bin/bash`
- `shell`

Możemy również wydać te dwie komendy już przy wywołaniu edytora *vim*. Służy do tego parametr `--cmd`, co ilustruje Rysunek 3. W efekcie, otrzymamy powłokę *bash*, bez restrykcji. Warto tutaj zaznaczyć, że *vim* pozwala na zablokowanie możliwości korzystania z powłoki. Wystarczy, że – podobnie jak w przypadku *basha* – utworzymy link symboliczny o nazwie *rvim*. Od tej pory, próba wykonania komendy `shell` skończy się komunikatem:

```
Shell commands not allowed in rvim
```




**Rysunek 2.** *Użycie funkcji shell escape w programach telnet, gdb oraz ftp*




**Rysunek 3.** *Uruchomienie powłoki /bin/bash przy pomocy funkcji shell edytora vim*

## Odwołania do pomocniczych programów

Niektóre programy użytkowe, mając na celu przeprowadzenie pewnych operacji, odwołują się do zewnętrznych aplikacji, np. popularny edytor *pico* odwołuje się do programu *ispell*, w celu sprawdzenia poprawności edytowanego tekstu, przeglądarki *links* oraz *lynx* wywołują program pocztowy, ilekroć otworzymy link zaczynający się od *mailto:* itd.

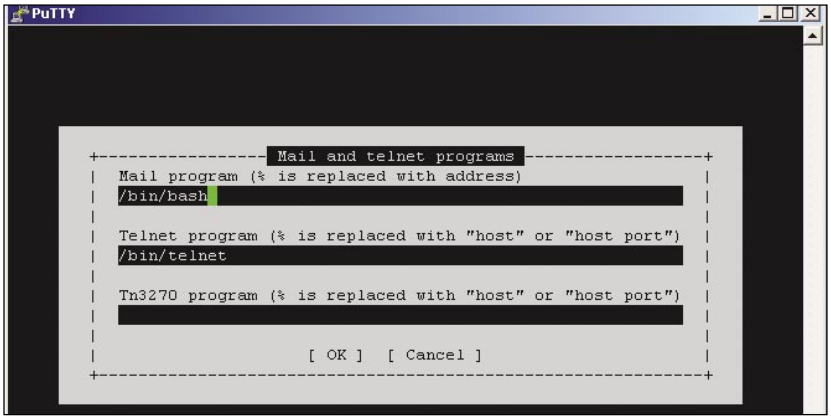
Problem pojawia się w momencie, gdy użytkownik ma możliwość określenia ścieżki do programu. Nic nie stoi wtedy na przeszkodzie, aby zamiast ścieżki do programu *ispell* – podać ścieżkę do zupełnie innej aplikacji. W zależności od sposobu, w jaki uruchamiany jest zewnętrzny program (w tym użyte przełączniki) – istnieje szansa na poprawne wykonanie wybranej przez nas aplikacji, którą może być program *bash*.

W przypadku programu *pico* sprawa jest prosta. Ścieżkę do programu *ispell* możemy zmieniać za pomocą przełącznika `-s`:

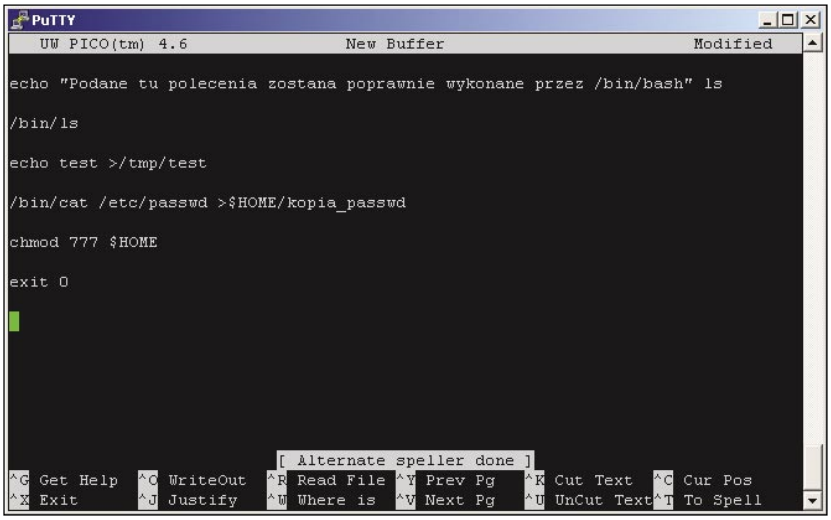
```
pico -s /bin/bash
```

Po uruchomieniu edytora, wpisujemy komendy, jakie chcemy przekazać do powłoki *bash*, oddzielając je znakami nowej linii. Edycję kończymy kombinacją klawiszy *[CTRL][T]*, która odpowiada za przekazanie bufora edycji do zdefiniowanego programu. Przekazanie to odbywa się poprzez zapis bufora do tymczasowego pliku, a dalej przekazanie ścieżki do niego jako argument:

```
/bin/bash /tmp/pico.xxxx
```




**Rysunek 4.** *Zmiana ścieżki programu obsługującego protokół mail w programie links*




**Rysunek 5.** *Wykorzystanie funkcji sprawdzającej pisownię w pico, do wykonania poleceń powłoki bash*

Efektem tak utworzonego polecenia, będzie zinterpretowanie tymczasowego pliku *pico* – zupełnie tak, jakby był on zwyczajnym skryptem *bash*.

Sporo programów pozwala na zdefiniowanie ulubionego edytora tekstu (np. *pine*). Nawet jeśli nie widać możliwości dokonania takiej zmiany w konfiguracji samego programu ani w dostępnych przełącznikach - należy sprawdzić, czy program nie korzysta ze zmiennej środowiskowej o nazwie `EDITOR` lub `VISUAL`. Zmienne te nie są chronione przez restrykcje *rbasha*, a zatem można przypisać im dowolną ścieżkę.

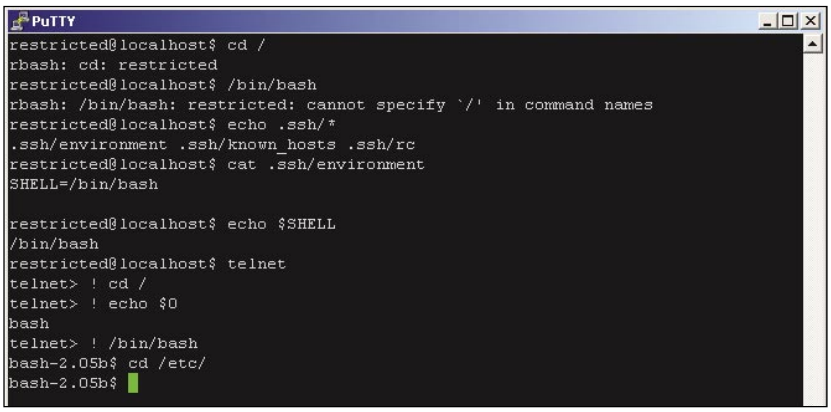
## Podmiana zmiennej $SHELL

Jeżeli na serwerze działa serwer SSH, a opcja `PermitUserEnvironment` znajdująca się w pliku konfiguracyjnym usługi *sshd_config* jest ustawiona na `ON` (opcja ta domyślnie jest wyłączona) – istnieje szansa na zmianę niektórych zmiennych środowiskowych, które mogą mieć poważny wpływ na działanie *rbasha*. Jest to możliwe, gdyż *deamon sshd* zaraz po uwierzytelnieniu użytkownika ustawia podstawowe zmienne środowiskowe, a następnie wczytuje plik *.ssh/environment*, jeśli takowy istnieje. Przypuśćmy, że w pliku tym znalazłaby się linijka:

```
SHELL=/bin/bash
```

Co się wtedy stanie? *Deamon SSH* po zalogowaniu użytkownika ustawi zmienną `SHELL` na wartość */bin/rbash* w oparciu o informacje z pliku *passwd*, jednak zaraz po odczycie pliku *environment* zmienna ulegnie nadpisaniu.

Do przeprowadzenia takiego ataku konieczne jest (oprócz wspomnianej konfiguracji *sshd*) istnienie katalogu *.ssh* (ewentualnie dostęp do polecenia `mkdir`) z możliwością zapisu. Gdy katalog domowy ma uprawnienia *+rx*, możemy w prosty sposób wylistować jego zawartość- używając (wbudowanej) komendy `echo *`. Konieczny będzie również dostęp do aplikacji, która pozwoli na utworzenie pliku *environment*. Nie musi to być edytor tekstu, równie dobrze w tej roli sprawdzi się popularny klient pocztowy *pine*.




**Rysunek 7.** *Ominięcie restrykcji poprzez nadpisanie $SHELL plikiem .ssh/environment*

Cała operacja sprowadza się do wysłania pliku *environment* mailem z innego konta, odebrania maila oraz skorzystania z funkcji programu *pine*, która umożliwia zapis treści maila w podanej lokalizacji. Funkcja ta jest dostępna po wybraniu opcji *ViewAttach*.

Jeśli zapis się powiedzie, po ponownym zalogowaniu na konto zmienna `SHELL` powinna zostać zamieniona. Dzięki temu, możliwe stanie się skorzystanie z – wcześniej omówionej – funkcji *shell escape*, któregoś z dostępnych programów, np. *telnet*, w celu wykonania poleceń w oparciu o zdefiniowaną przez nas powłokę.

## .forward

Plik *.forward* umieszczony w katalogu domowym użytkownika używany jest przez program *sendmail* w celu przekierowania maili (przeznaczonych dla użytkownika) pod wskazany adres. Oprócz adresów mailowych, dozwolone jest podanie ścieżki do programu, do którego chcemy przesłać treść listu. Posiadając możliwość edycji tego pliku, ustawiamy go na:

```
john@mailserver.com, "| /bin/bash -c
                        nasza_komenda"
```

Nadchodzący mail zostanie wysłany pod adres Johna, ale także na wejście programu *bash* - powodując wykonanie naszej komendy.

Na szczęście taki atak nie powiedzie się w większości przypadków – ze względu na to, że podane polecenie najczęściej zostaje wykonane w oparciu o specjalną restrykcyjną powłokę *sendmaila* zwaną *smrsh*. Powłoka ta sprawia, że wykonane mogą zostać jedynie programy zawarte w katalogu */etc/smrsh*.

## Zmiana uprawnień plików i katalogów

Jak już zdążyliśmy się przekonać – uprawnienia nadane plikom czy katalogom zawartym w katalogu domowym mają ogromne znaczenie. Jeżeli administrator omyłkowo udostępni komendę `chmod`, istnieje ryzyko, że użytkownik nada katalogowi domowemu prawo *+w*, co uprawni go do skasowania pliku *.bash_profile*. Przy kolejnym zalogowaniu, zmienna `PATH` nie zostanie ograniczona do */usr/local/rbin* – a co za tym idzie, istnieje duże prawdopodobieństwo, że w ścieżce znajdzie się katalog */bin* (a to już tylko krok od uruchomienia normalnego *basha*).

Czasami istnieje możliwość zmiany uprawnień bez użycia programu */bin/chmod*. Wyobraźmy sobie, że na tym samym serwerze działa usługa FTP. O ile administrator nie ograniczył dostępu do tej usługi, a także nie wprowadził restrykcji na instrukcję FTP SITE CHMOD, użytkownik po połączeniu na port 21 może wydać komendę `chmod 777`, aby dalej bez większych przeszkód wykasować plik *.bash_profile* lub też nadpisać go innym.

## Przerwanie skryptu startowego

Niejednokrotnie administratorzy tworząc rozbudowane skrypty startowe zapominają o obsłudze sygnałów. Spójrzmy chociażby na skrypt zawarty w Listingu 1.

Wystarczy, że użytkownik w czasie działania komendy `sleep 5`, wciśnie kombinację klawiszy *[CTR-L][C]* – powodując wysłanie sygnału `SIGINT`. Poskutkuje to natychmiastowym zakończeniem skryptu. Jeżeli wykonanie komendy `export` nie dojdzie do skutku, zmienna `PATH` nie zostanie ustawiona, a tym samym użytkownik nie zostanie ograniczony do komend z katalogu *rbin*. Sygnały mogą być kontrolowane za pomocą funkcji `trap`. Zignorowanie sygnału `SIGINT` można uzyskać komendą:

```
trap "" SIGINT
```

**Listing 1.** *Nieco bardziej rozbudowany skrypt .bash_profile*

```
echo -n "Witamy na serwerze"
echo `/bin/hostname`
echo
echo "Prosimy zapoznac sie z zasadami: "
cat /etc/motd
sleep 5
clear
export PATH=/usr/local/rbin
enable -n pwd
```

**Listing 2.** *Współdzielona biblioteka, uruchamiająca powłokę bash*

```
#include <stdio.h>
void localtime()
{
   unsetenv("LD_PRELOAD");
   system("echo ok! && /bin/bash");
   exit(0);
}
```

## Załadowanie biblioteki

Bardzo ciekawą – a zarazem dającą duże szanse powodzenia – metodą na wydostanie się z ograniczonej powłoki, jest załadowanie wcześniej utworzonej, specjalnie spreparowanej biblioteki. Jest to możliwe, gdyż systemowy *linker*/*loader* (*ld.so*) pozwala na określenie współdzielonej biblioteki, która zostanie załadowana przed wykonaniem programu. Daje to z kolei szansę na nadpisanie wybranej funkcji, używanej przez uruchamiany (przez *loader*) program.

Ścieżkę do biblioteki, jaką ma dla nas załadować *linker*, możemy wskazać na dwa sposoby - zapisując ją w pliku */etc/ld.so.preload* lub w zmiennej środowiskowej `LD_PRELOAD`. Dostęp do pliku *ld.so.preload* ma tylko *root* – dlatego pozostaje posłużenie się zmienną. Powłoka *rbash* nie ustawia tej zmiennej w tryb *readonly*. O ile administrator nie zablokował wszystkich komend służących do zmiany środowiska (co się rzadko zdarza), będziemy w stanie przypisać do niej dowolną ścieżkę.

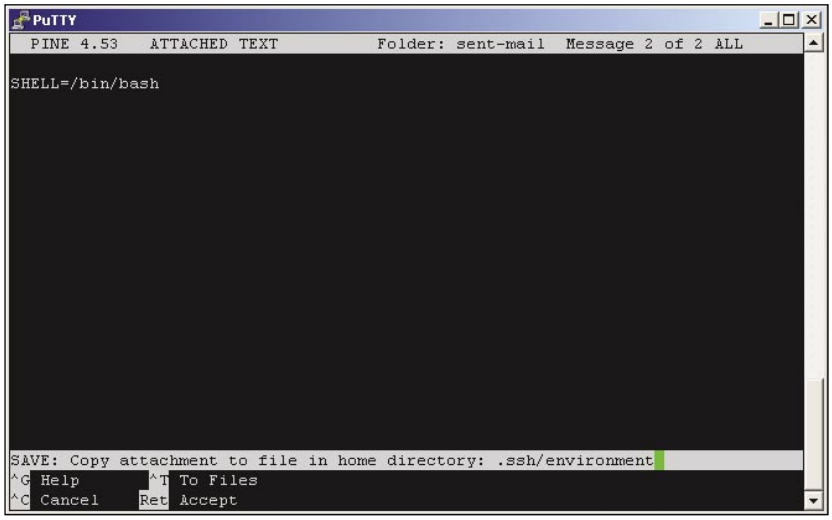
Oprócz dostępu do polecenia zmieniającego środowisko, potrzebne będzie miejsce oraz sposób, w jaki zapiszemy plik binarny biblioteki. Jeśli mamy prawo zapisu do katalogu domowego (lub jednego z podkatalogów) oraz dostęp do FTP – problem rozwiązany. Gdy mamy dostęp do programu *pine*, skompilowaną bibliotekę możemy przesłać w załączniku maila. Program *pine* nie nadaje prawa do wykonywania zachowywanym załącznikom – co jednak nie przeszkadza w załadowaniu biblioteki dynamicznej przez *ld.so* (wystarczy, że plik posiada prawo *+r*) . Pozostaje problem miejsca, w którym zapiszemy plik. Nawet jeśli nie posiadamy prawa *+w* w obrębie całego katalogu domowego, wciąż mamy szansę na powodzenie naszych działań. Możemy wykorzystać jeden z katalogów na pliki tymczasowe, */tmp*, lub */var/tmp* (nawet jeśli jest on zamontowany jako *noexec*). Załóżmy jednak najbardziej restrykcyjny scenariusz – nie mamy dostępu do FTP, prawa zapisu w katalogu domowym, a jedyny przyznany nam program to prosty kalendarzyk – *cal*. Okazuje się, że i z tej sytuacji jest wyjście. Skoro do systemu zalogowaliśmy się poprzez SSH, najpewniej uda się skopiować nasz binarny plik do katalogu */tmp* przy pomocy *scp*.

Pozostaje napisanie współdzielonej biblioteki. Jako że atak będzie polegał na nadpisaniu jednej z funkcji atakowanego programu naszym kodem – zanim przystąpimy do tworzenia biblioteki, musimy ustalić, z jakich funkcji on korzysta. Trzymając się założonego scenariusza, przystępujemy do badania jedynego programu, do którego mamy dostęp – *cal*. Aby poznać używane przez ten program funkcje, możemy wyświetlić tablicę symboli za pomocą programu *objdump*, w następujący sposób:

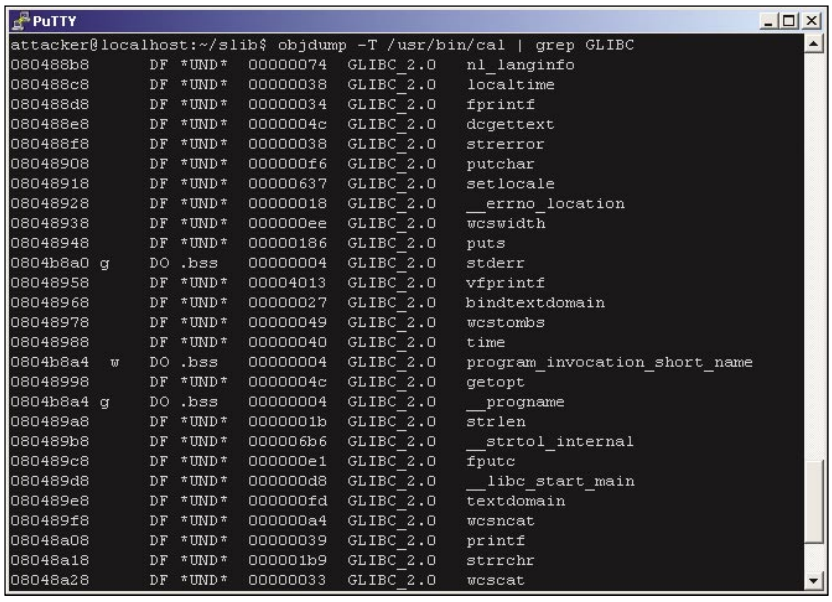
```
objdump -T /usr/bin/cal | grep GLIBC
```

Widzimy, że jedną z wykorzystywanych funkcji, jest funkcja `localtime()`. Posiadając tą informację, możemy przystąpić do pisania programu biblioteki.

Przedstawiony na Listingu 2 przykładowy kod języka C wykonuje cztery czynności. A mianowicie: usunięcie zmiennej `LD_PRELOAD` ze środowiska (w celu uniknięcia ingerencji w wykonywane dalej programy), wyświetlenie tekstu *ok!*, uruchomienie programu powłoki */bin/bash* oraz




**Rysunek 6.** *Wykorzystanie programu pine do zapisu pliku environment*




**Rysunek 8.** *Tablica symboli programu cal*

przerwanie działania programu (*cal*), zaraz po zakończeniu pracy z powłoką. Kompilację należy przeprowadzić w taki oto sposób:

```
gcc -shared slibrary.c -o slibrary
```

Po przesłaniu biblioteki na serwer, jesteśmy gotowi do przeprowadzenia właściwej części ataku. Po zalogowaniu, ustawiamy zmienną środowiskową `LD_PRELOAD` komendą `declare` lub `export`. Zakładamy tutaj, że biblioteka znajduje się w katalogu *tmp*:

```
declare -x LD_PRELOAD=/tmp/slibrary
```

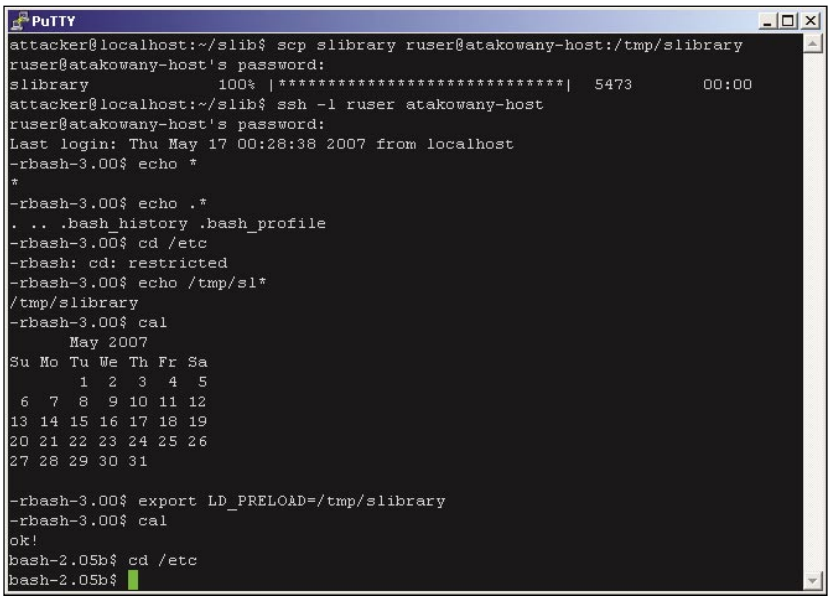
Następnie uruchamiamy kalendarz, pisząc po prostu `cal`. Jeśli wszystko się powiedzie - program, w którymś momencie swojego działania, odwoła się do funkcji `localtime()`, pod którą znajdzie się nasz kod. Powinniśmy ujrzeć napis *ok!* oraz znak zachęty nowej powłoki.

## Jak się zabezpieczyć?

Przedstawione przykłady ataków pokazują, jak łatwo jeden źle dobrany program (oferujący pozornie nieistotną funkcję, jak ustawienia ulubionego edytora) może doprowadzić do wydostania się z restrykcyjnej powłoki *rbash*. Czy można w jakiś skuteczny sposób sprawić, aby restrykcyjne środowisko stało się mniej zawodne? Z pomocą przychodzą dwa projekty *open source*.

Pierwszy z nich – o nazwie *jail* służy do utworzenia odseparowanego środowiska, w którym zamykany zostaje użytkownik zaraz po zalogowaniu. Środowisko to tworzone jest dzięki wykorzystaniu funkcji `chroot()`. Proces działający w środowisku *chroot* ograniczony jest do określonego katalogu i nie ma dostępu do głównego drzewa. Taki proces postrzega katalog, w którym został zamknięty, jako katalog główny – *root* (*/*). Uzyskujemy w ten sposób zwiększone bezpieczeństwo, gdyż użytkownik – nie mogąc wyjść poza przydzielone procesowi drzewo – ma dostęp jedynie do plików utworzonych w jego obrębie. Tyczy się to wszystkich plików, łącznie z bibliotekami, programami i plikami urządzeń. Projekt *jail* znacznie ułatwia zgromadzenie wszystkich plików, potrzebnych do działania takiego środowiska.

Drugi projekt nosi nazwę *Iron Bars Shell – restricted system shell for Linux/Unix*. Jest to projekt restrykcyjnej powłoki, nazywanej w skrócie *ibsh*. Jest ona tworzona z myślą o bezpieczeństwie. Jedną z głównych idei projektu jest zakazanie wszystkich operacji, które nie zostały jawnie określone jako dozwolone. Powłoka oferuje też kilka ciekawych funkcji, jak kontrola parametrów wykonywanych poleceń, logowanie poczynań użytkownika do *sysloga* czy też blokowanie dostępu do plików, których rozszerzenie nie znajduje się na liście dozwolonych rozszerzeń utworzonej przez administratora (ustawienie blokady na pliki **.c*, zapobiegnie kompilacji potencjalnie szkodliwych źródeł C itp.). W ten sposób zyskujemy dużo większą kontrolę nad poczynaniami użytkownika.




**Rysunek 9.** *Wyjście z restrykcyjnego środowiska, dzięki załadowaniu podstawionej biblioteki*

## Podsumowanie

Praktyka pokazuje, że ciężko jest utworzyć restrykcyjne środowisko, bazujące tylko na restrykcyjnej powłoce. Trzeba przewidzieć każdy ruch, jaki może wykonać użytkownik oraz solidnie zgłębić zasady działania każdego z udostępnianych programów, pod kątem oferowanych przezeń funkcji. Dobranie bezpiecznych programów może stanowić niemały kłopot, gdyż, jak się przekonaliśmy, nawet prosty edytor tekstu potrafi zawierać funkcje pozwalające na dostęp do powłoki. Możemy dojść do wniosku, że powłoka restrykcyjna powinna być stosowana tylko jako uzupełnienie bardziej efektywnych technik ograniczania użytkowników (jak *chroot*) lub ewentualnie – jako powłoka dla początkujących użytkowników systemu. ●


### O autorze

Autor jest samoukiem, pasjonatem, od wielu lat interesującym się Informatyką, a w szczególności aspektami bezpieczeństwa. Studiuje sieci komputerowe w ramach programu *Cisco Network Academy* na Politechnice Poznańskiej. Kontakt z autorem: *golunski@crackpl.com*


### W Sieci

- *http://www.jmcresearch.com/projects/jail/* – strona domowa projektu *Jail*,
- *http://ibsh.sourceforge.net* – strona domowa projektu *Iron Bars Shell*.