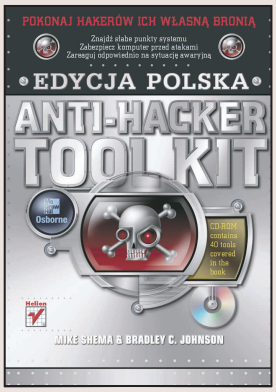
# Anti-Hacker Tool Kit. Edycja polska




Autorzy: Mike Shema, Bradley C. Johnson
Tłumaczenie: Andrzej Grażyński, Szymon Kobalczyk, Mikołaj Szczepaniak
ISBN: 83-7361-478-8
Tytuł oryginału: Anti-Hacker Tool Kit
Format: B5, stron: 808

## Pokonaj hakerów ich własną bronią

Zagrożenie ze strony hakerów to już nie temat książek i filmów fantastycznych, ale rzeczywistość. Hakerzy atakują nie tylko duże firmy i organizacje, ale także sieci lokalne i pojedyncze komputery połączone z internetem. Wiedza o metodach ich działania i sposobach zapobiegania włamaniom jest niezbędna każdemu administratorowi sieci. Oczywiście, sama wiedza to za mało – trzeba jeszcze umieć ją wykorzystać w praktyce.

Książka „Anti-Hacker Tool Kit. Edycja polska" zawiera omówienie narzędzi służących do zabezpieczania komputerów i sieci. Najważniejsze jest jednak to, że poza opisami narzędzi znajdziesz w niej również informację o tym, kiedy i dlaczego należy ich użyć. Dowiesz się, jak konfigurować i stosować skanery portów, łamacze haseł, narzędzia do kontroli systemów, skanery słabych punktów usług WWW, szperacze oraz narzędzia do usuwania i dokumentowania skutków włamań. Nauczysz się wykorzystywać w walce z hakerami te same techniki, które stosują oni podczas włamań. Dzięki opisywanym w książce narzędziom i technikom wykryjesz słabe punkty komputera i sieci, zlikwidujesz je, a w razie ataku – odpowiednio na niego zareagujesz.

- Skanery portów
- Narzędzia do enumeracji
- Narzędzia do łamania haseł
- Narzędzia do testowania systemów
- Firewalle
- Badanie sieci bezprzewodowych
- Narzędzia do tworzenia kopii dowodowej

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

# Spis treści

Rozdział 8.

# Łamanie haseł: narzędzia testujące wszystkie możliwości

Uśmiech, klucze, hasło. Niezależnie od tego, czy chcesz się dostać do nocnego klubu, do swojego domu, czy do komputera, zawsze potrzebujesz czegoś, czego nie ma nikt poza Tobą. Hasła użytkowników sieci komputerowej powinny być na tyle trudne do odgadnięcia, by żaden użytkownik nie mógł się domyślić, jakie jest hasło jego kolegi siedzącego przy sąsiednim biurku (zakładamy przy tym, że kolega nie trzyma tego hasła na kartce ukrytej pod klawiaturą). Wadą takiego rozwiązania jest fakt, że odkrycie przez hakera choćby jednego hasła może sprawić, że najlepsze zabezpieczenia komputerów, najnowsze aktualizacje systemów operacyjnych i najbardziej restrykcyjne reguły firewalla nie będą dla niego przeszkodą.

Haker próbujący ustalić hasło ma generalnie do wyboru dwie strategie. Może spróbować przechwycić przesyłane hasło i, jeśli transmisja była szyfrowana, wykorzystać narzędzia testujące wszystkie możliwości do jego odszyfrowania. Haker może także podjąć próbę odgadnięcia hasła. Łamanie haseł jest jedną z najstarszych technik wykorzystywanych przez hakerów, a jej skuteczność wynika przede wszystkim z faktu, że ludzie nie są zbyt dobrymi generatorami liczb losowych.

Ważne jest, byś dobrze rozumiał, w jaki sposób (i gdzie) przechowywana jest większość haseł, dzięki czemu będziesz wiedział, jak działają narzędzia do łamania haseł i z jakich korzystają metod. Hasła w systemach Unix i Windows są przechowywane w postaci jednostronnie zakodowanej, co oznacza, że nie można ich odszyfrować. Zamiast tego, logowanie użytkownika polega na zaszyfrowaniu podanego przez niego ciągu znaków i porównaniu go z ciągiem przechowywanym w pliku haseł. Przykładowo, hasło użytkownika *abc123* jest przechowywane w systemie Unix w postaci łańcucha *kUge2g0BqUb7k* (pamiętaj, że nie możemy odszyfrować tego ciągu). Wyobraźmy sobie, że kiedy użytkownik ten próbuje się zalogować w systemie popełnia błąd i wprowadza ciąg znaków *abc124*. System Unix wywołuje funkcję `crypt()` dla podanego hasła, aby wygenerować tymczasowy zaszyfrowany łańcuch. Taki łańcuch dla hasła *abc124* będzie się różnił od przechowywanego łańcucha dla hasła *abc123*, zatem system poinformuje użytkownika o tym, że podane przez niego hasło jest nieprawidłowe. Zauważ, że podane hasło (*abc124*) jest szyfrowane i porównywane z przechowywanym zakodowanym łańcuchem (*kUge2g0BqUb7k*).

Łańcuch ten nie jest odszyfrowywany. Najprostsza metoda ataku, którego celem jest złamanie hasła, polega na stworzeniu zaszyfrowanych łańcuchów dla znanych słów i porównywaniu ich z docelowymi zakodowanymi łańcuchami dla rzeczywistych haseł.

# Narzędzie PassFilt.dll i strategie zarządzania hasłami w systemie Windows

System Windows NT 4.0 udostępnia metodę wymuszania tworzenia przez użytkowników skomplikowanych haseł. Narzędzie PassFilt.dll, które zostało wprowadzone w pakiecie Service Pack 2, umożliwia administratorom ustanawianie elementarnych reguł dla haseł użytkowników. Implementacja reguł konstruowania haseł jest w wielu przypadkach dobrym sposobem zabezpieczenia systemu. Możemy przecież instalować najnowsze aktualizacje zabezpieczeń i stworzyć najsurowszą konfigurację serwera, musimy jednak pamiętać, że jedno słabe hasło może narazić nasz system na skuteczny atak.

## Implementacja

Biblioteka PassFilt.dll może znajdować się w systemie NT, zanim jednak narzędzie to zacznie prawidłowo funkcjonować, musimy wprowadzić kilka niezbędnych modyfikacji w Rejestrze:

1. Upewnij się, że plik *PassFilt.dll* znajduje się w katalogu *C:\WINNT\System32* (lub dowolnym innym katalogu *%SYSTEMROOT%*).
2. Użyj edytora Rejestru (w tym przypadku program *regedt32.exe* sprawdza się lepiej niż *regedit.exe*) do otwarcia lokalizacji *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa*.
3. W prawym panelu zaznacz (kliknij lewym przyciskiem myszy) klucz *Notification Packages*.
4. Z menu *Edycja* wybierz opcję *Modyfikuj*.
5. Jeśli już istnieją dane wartości dla *FPNWCLNT*, usuń je, chyba że w Twoim systemie wymagana jest zgodność z systemem Novell.
6. Wpisz wartość ***passfilt***.


Pamiętaj, że jeśli zastosujesz narzędzie PassFilt.dll dla głównego kontrolera domeny, powinieneś zastosować to narzędzie także dla zapasowych kontrolerów.

Jeśli wszystko poszło zgodnie z planem, każda zmiana hasła dokonywana od tego momentu przez każdego użytkownika *z wyjątkiem* administratora będzie musiała spełniać określone reguły. Ograniczenia narzucone przez narzędzie PassFilt.dll są jednak tylko niewielkim krokiem na drodze do naprawdę dobrych haseł. System Windows będzie od tej pory sprawdzał każde nowe hasło pod kątem zgodności z następującymi regułami:

- Hasło nie może być częścią nazwy konta użytkownika.
- Hasło musi się składać z co najmniej sześciu znaków.

- Musi zawierać znaki z trzech z poniższych kategorii:
  - wielkie litery (od A do Z),
  - małe litery (od a do z),
  - cyfry (od 0 do 9),
  - znaki niealfanumeryczne (np. interpunkcyjne).

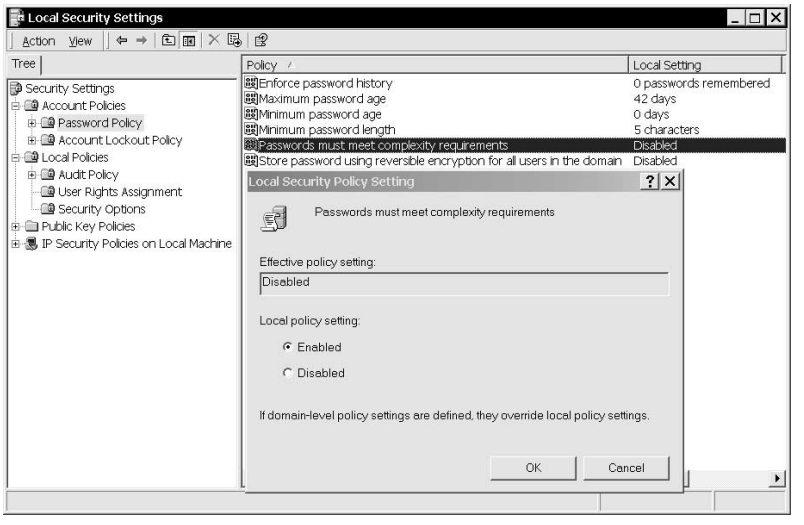
Powyższych reguł nie można zmodyfikować. Z technicznego punktu widzenia, biblioteka dołączana dynamicznie (ang. *Dynamic-Link Library — DLL*) może oczywiście zostać zastąpiona napisanym przez nas plikiem zgodnym z odpowiednim interfejsem API systemu Windows, jednak — jak się przekonamy w podrozdziale „L0phtCrack" — schemat szyfrowania systemu Windows NT może okaleczyć nawet „najmocniejsze" hasła. Narzędzie PassFilt.dll nie powinno więc być traktowane jak panaceum na wszystkie problemy związane z hasłami w systemie Windows. Użytkownik nadal może stworzyć słabe hasło, które będzie łatwe do odgadnięcia dla łamacza haseł lub dobrego skryptu testującego wszystkie możliwości. Rozważmy poniższe przykłady słabych haseł, które spełniają warunki stawiane przez narzędzie PassFilt.dll:

- Passw0rd
- Password!
- p4ssw0rd!
- Pa55werd

Wygląda na to (przynajmniej na podstawie przebadanych przez nas ponad dziesięciu tysięcy złamanych haseł), że użytkownicy lubią zastępować samogłoski cyframi (*4* zamiast *a*, *3* zamiast *e*, *1* zamiast *i*, *0* zamiast *o*) i często dodają na końcu swoich haseł wykrzykniki, aby obejść tego typu ograniczenia i stworzyć „dobre" hasła. Dobre słowniki haseł, które są w posiadaniu większości łamaczy haseł, zawierają permutacje najczęściej spotykanych słów składających się z liter i symboli. Hasła oparte na nazwach ulubionych klubów sportowych, miast, na przekleństwach i imionach są zawsze najłatwiejsze do złamania, ponieważ hakerzy doskonale zdają sobie sprawę z tego, że użytkownicy robią wszystko, by ich hasła nie były zbyt trudne do zapamiętania. Powinniśmy więc poświęcać wiele uwagi ochronie listy haseł oraz usług (w tym poczty elektronicznej, Secure Shell, Windows NetBIOS) bazujących na hasłach.

## Zasady zabezpieczeń lokalnych w systemach Windows 2000 i Windows XP

W systemach wydanych po serii NT oddzielono ustawienia narzędzia PassFilt.dll od Rejestru i stworzono dla nich specjalny graficzny interfejs użytkownika (GUI). Nie wprowadzono jednak ani dodatkowych reguł, ani metod modyfikowania już istniejących. Aby włączyć wymuszanie złożonych haseł, kliknij dwukrotnie ikonę *Local Security Policy* (*Zasady zabezpieczeń lokalnych*) w folderze *Administrative Tools* (*Narzędzia administracyjne*) dostępnym w *Control Panel* (*Panel sterowania*). Okno *Local Security Settings* (*Ustawienia zabezpieczeń lokalnych*) przedstawiono na rysunku 8.1.




**Rysunek 8.1.** *Zwiększanie złożoności hasła*

# PAM oraz strategie zarządzania hasłami w systemach Unix

Część popularnych dystrybucji systemu Unix, w tym FreeBSD, Linux i Solaris, zawiera tzw. moduł PAM (skrót pochodzi od ang. *Pluggable Authentication Module* i nie ma nic wspólnego z funkcją PAM oprogramowania ISS). Moduł PAM kontroluje wszystkie działania, które wymagają od użytkownika podania hasła. Może to być próba uzyskania dostępu do usługi telnet, logowanie do konsoli lub zmiana istniejącego hasła. Implementacje PAM są dostępne także dla mocniejszych schematów uwierzytelniania, w tym schematu Kerberos, S/Key oraz RADIUS. Konfiguracja modułu pozostaje niezmieniona niezależnie od metody uwierzytelniania czy aplikacji wymagającej uwierzytelnienia. Skupmy się więc na sposobach wymuszania strategii zarządzania hasłami za pomocą modułu PAM.

## Implementacja dla systemu Linux

Zobaczmy, jak możemy ustawić zasady bezpieczeństwa w systemie Linux i porównajmy otrzymane rezultaty z narzędziem PassFilt.dll dostępnym w systemach Windows NT. W pierwszej kolejności musimy się upewnić, że w naszej dystrybucji jest zainstalowana biblioteka cracklib (lub libcrack). Ta opracowana przez Aleca Muffeta biblioteka sprawdzająca hasła jest częścią domyślnych instalacji dystrybucji Debian, Mandrake, Red Hat i SuSE. Aby zaimplementować sprawdzanie haseł, musimy jedynie zmodyfikować plik tekstowy zawierający konfigurację modułu PAM. Może to być jeden z dwóch plików: */etc/pam.conf* lub */etc/pam.d/passwd*.

Wpis w pliku */etc/pam.conf* związany ze zmianami haseł powinien wyglądać podobnie do poniższego:

```
passwd  password required       /lib/security/pam_cracklib.so retry=3
passwd  password required       /lib/security/pam_unix.so nullok use_authtok
```

Plik ten jest logicznie podzielony na pięć kolumn. Pierwsza z nich zawiera nazwę usługi, czyli nazwę programu, na którego działanie mają wpływ instrukcje zdefiniowane w pozostałych kolumnach. Plik */etc/pam.d/passwd* zawiera tylko cztery kolumny, ponieważ jego nazwę określa usługa `passwd`. Ten styl konfiguracji po prostu rozdziela nazwy usług pomiędzy różne pliki, zamiast stosować np. jednolity plik dla wielu usług. Jednak niezależnie od stylu konfiguracji, dla jednej usługi może istnieć wiele wpisów. Taki model nosi nazwę *kaskadowo ułożonych modułów* dla jednej usługi.

Oto przykład pliku */etc/pam.d/passwd* z tak ułożonymi modułami:

```
password required        /lib/security/pam_cracklib.so retry=3
password required        /lib/security/pam_unix.so nullok use_authtok
```

Pierwsza kolumna oznacza typ modułu, do którego odnosi się dany wpis. Kolumna ta może zawierać jeden z czterech typów (nas interesuje przede wszystkim modyfikacja typu modułu kontrolującego zmiany haseł):

- `account` — usługa kontroluje działania aplikacji na podstawie atrybutów użytkownika (czyli konta, z którego ten użytkownik korzysta), sprawdza np. czy użytkownik ma prawa odczytu dla danego pliku. Przykładowo, możemy użyć wpisu dla konta, aby umożliwić użytkownikowi dostęp do pewnych zasobów, takich jak udostępniane pliki. Pamiętaj jednak, że bez wpisu `auth` użytkownik nie będzie mógł się zalogować w systemie.
- `auth` — odpowiada za komunikację typu wezwanie-odpowiedź z użytkownikiem, w tym wysyłanie do użytkownika monitów o podanie hasła. Ta usługa jest używana za każdym razem, gdy system lub zasób zezwala użytkownikowi na zalogowanie się.
- `password` — aktualizuje informacje uwierzytelniające, np. zmianę hasła użytkownika. Wpisy tego typu nie mają nic wspólnego z samym logowaniem użytkowników w systemie. Jedynym ich zadaniem jest zezwalanie na dostęp do systemu bezpieczeństwa użytkownika kontrolującego dane uwierzytelniające.
- `session` — obsługuje działania mające miejsce przed skorzystaniem z usługi lub po nim, takie jak kontrolowanie prób logowania zakończonych niepowodzeniem. Przykładowo, możemy użyć takiego wpisu do wyświetlania czasu i daty natychmiast po zalogowaniu użytkownika w systemie. W takim przypadku pierwszy wpis (`auth`) dotyczyłby weryfikacji hasła podanego przez użytkownika, natomiast drugi wpis (`session`) wywoływałby moduł PAM w celu wyświetlenia aktualnego czasu. Innym zastosowaniem wpisu typu session mogłoby być wykonywanie konkretnych funkcji w momencie, gdy użytkownik wylogowuje się z systemu, np. wyświetlenie odpowiedniego wpisu z dziennika zdarzeń lub pozbawienie ważności tymczasowego identyfikatora.

Kolejna kolumna dotyczy *sterowania* usługą, czyli określa sposób, w jaki usługa powinna być obsługiwana. Pomyślne wykonanie usługi oznacza, że wykonuje ona pewną funkcję, np. zmieniającą hasło użytkownika. Uruchomienie zakończone niepowodzeniem oznacza, że usługa nie otrzymała prawidłowych danych, np. hasła użytkownika. Poniżej przedstawiamy dopuszczalne metody sterowania:

- `requisite` — jeśli próba wykonania usługi zakończyła się niepowodzeniem, wszystkie kolejne działania (kaskadowo połączone usługi) automatycznie zakończą się niepowodzeniem. Oznacza to, że żadna kolejna usługa na stosie nie zostanie pomyślnie wykonana.
- `required` — jeśli próba wykonania usługi zakończyła się niepowodzeniem, proces jest kontynuowany, ale ostatecznie kończy się niepowodzeniem. Jeśli na stosie znajdują się inne działania, nie można wykluczyć, że zakończą się pomyślnie, jednak nie zmieni to całościowego wyniku.
- `optional` — jeśli usługa zakończy się sukcesem lub niepowodzeniem, proces jest kontynuowany. Wynik wykonania tego działania w żaden sposób nie wpływa na wyniki wykonania pozostałych usług na tym stosie.
- `sufficient` — jeśli usługa zakończy się sukcesem i żadne kroki konieczne (`requisite`) ani wymagane (`required`) nie zakończą się niepowodzeniem, proces jest przerywany i kończy się sukcesem.

Kolejna kolumna zawiera *ścieżkę modułu* wykorzystywanej biblioteki uwierzytelniania. Ścieżka modułu powinna zawierać pełną ścieżkę dostępu i nazwę pliku biblioteki obsługującej uwierzytelnianie. Będziemy używali biblioteki cracklib, zatem musimy się upewnić, że zawartość tej kolumny wskazuje na plik `pam_cracklib.so`.

Ostatnia kolumna zawiera argumenty przekazywane do biblioteki obsługującej uwierzytelnianie. Jeśli raz jeszcze rzucimy okiem na pierwszy przykład pliku */etc/pam.conf*, zobaczymy, że działanie modułu `pam_cracklib.so` musi zakończyć się sukcesem z argumentem `retry=3` dla użytkowników zmieniających swoje hasła za pomocą programu `passwd`:

```
passwd  password required       /lib/security/pam_cracklib.so retry=3
```

## Argumenty biblioteki cracklib

Biblioteka cracklib oferuje w rzeczywistości znacznie więcej argumentów, nie tylko proste `retry=`*n*. Argument `retry` jedynie instruuje program `passwd`, ile razy ma być przedstawiane użytkownikowi wezwanie do wprowadzenia nowego hasła. Sukces lub niepowodzenie usługi wykorzystującej moduł `pam_cracklib.so` zależy od liczby „punktów" zebranych przez użytkownika. Użytkownik zbiera punkty na podstawie zawartości proponowanego hasła. Argumenty modułu określają liczbę punktów przyznawanych za konkretne składniki nowego hasła:

- `minlen=`*n* (domyślnie *n* = 9) — minimalna długość, czyli minimalna liczba przyznanych punktów. Jeden punkt jest równoważny z jedną jednostką długości. Rzeczywista długość nowego hasła nie może być mniejsza niż 6, nawet mimo punktów przyznanych za złożoność tego hasła.
- `dcredit=`*n* (domyślnie *n* = 1) — maksymalna liczba punktów za umieszczenie w haśle cyfr (0 – 9). Jeden punkt jest przyznawany za jedną cyfrę.
- `lcredit=`*n* (domyślnie *n* = 1) — maksymalna liczba punktów za umieszczenie w haśle małych liter. Jeden punkt jest przyznawany za jedną literę.

- `ucredit=n` (domyślnie *n* = 1) — maksymalna liczba punktów za umieszczenie w haśle wielkich liter. Jeden punkt jest przyznawany za jedną literę.
- `ocredit=n` (domyślnie *n* = 1) — maksymalna liczba punktów za umieszczenie w haśle znaków nie będących literami ani cyframi. Jeden punkt jest przyznawany za jeden znak.

Pięć pozostałych argumentów nie wpływa bezpośrednio na punktację użytkownika:

- `debug` — rejestruje informacje diagnostyczne na podstawie ustawień dziennika systemowego.
- `difok=n` (domyślnie *n* = 10) — liczba znaków użytych w starym haśle, które nie mogą się powtarzać w nowym haśle. Jeśli przynajmniej połowa znaków w nowym haśle jest inna niż w poprzednim haśle, opcja jest ignorowana.
- `retry=n` (domyślnie *n* = 1) — liczba podejmowanych prób uzyskania od użytkownika poprawnego nowego hasła w przypadku, gdy poprzednie hasło nie spełniało warunku `minlen`.
- `type=text` — tekst zastępujący domyślne monity systemu Unix: „New UNIX password" i „Retype UNIX password".
- `use_authtok` — używana opcja kaskadowego łączenia modułów w usłudze. Jeśli użyjemy tej opcji, aktualny moduł będzie używał danych wejściowych modułu zdefiniowanego bezpośrednio przed nim w pliku konfiguracji, zamiast żądać od użytkownika ponownego wprowadzenia tych danych. Użycie tej opcji może być niezbędne, jeśli moduł cracklib nie został umieszczony na szczycie stosu.

Argumenty są umieszczane w ostatniej kolumnie wiersza i są oddzielone spacjami. Przykładowo, nasz administrator chce, by jego użytkownicy tworzyli hasła 15-znakowe, ale także, by za stosowanie cyfr przyznawane były maksymalnie dwa punkty i tyle samo punktów można było maksymalnie uzyskać za inne (niż cyfry i litery) znaki. W takim przypadku administrator powinien w pliku */etc/pam.d/passwd* umieścić następujące dwa wiersze:

```
password  required /lib/security/pam_cracklib.so minlen=15  dcredit=2 ocredit=2
password  required /lib/security/pam_unix.so nullok use_authtok md5
```

Zwróć uwagę na fakt dodania argumentu `md5` do biblioteki `pam_unix.so`. W ten sposób administrator włączył szyfrowanie haseł za pomocą algorytmu MD5. Hasła szyfrowane za pomocą używanego domyślnie algorytmu Data Encryption Standard (DES) nie mogą mieć więcej niż osiem znaków. Nawet z tak hojnie przyznawanymi punktami stworzenie 15-punktowego hasła z ośmiu znaków byłoby bardzo trudne! Hasła szyfrowane za pomocą algorytmu MD5 mogą mieć nieograniczoną długość.

Przyjrzyjmy się teraz kilku przykładowym prawidłowym i nieprawidłowym hasłom (wraz z odpowiadającymi im liczbami przyznawanych punktów) weryfikowanym przez nowy plik */etc/pam.d/passwd*. Pamiętaj, że domyślną wartością argumentów `lcredit` i `ucredit` jest 1:

| | |
|---|---|
| password | 9 punktów (8 punktów za długość + 1 punkt za małe litery) |
| passw0rd! | 12 punktów (9 punktów za długość + 1 punkt za małe litery + 1 punkt za cyfrę + 1 punkt za inny znak) |

| | |
|---|---|
| Passw0rd! | 13 punktów (9 punktów za długość + 1 punkt za wielką literę + 1 punkt za małe litery + 1 punkt za cyfrę + 1 punkt za inny znak) |
| Pa$$w00rd | 15 punktów (9 punktów za długość + 1 punkt za wielką literę + 1 punkt za małe litery + 2 punkty za cyfry + 2 punkty za inne znaki) |

Jak widać, samo ustalenie dużej wartości argumentu `minlen` może wymuszać od użytkowników definiowanie dosyć skomplikowanych haseł. Dwanaście punktów to prawdopodobnie najniższy próg, jaki powinniśmy zdefiniować w naszym systemie, w którym piętnaście punktów jest górną granicą. W innym przypadku użytkownicy byliby zmuszeni do zapisywania swoich haseł i trzymania ich obok klawiatury, a to w oczywisty sposób naraziłoby nasz system na niebezpieczeństwo.

# Plik login.conf w systemie OpenBSD

W systemie operacyjnym OpenBSD algorytm szyfrujący DES jest wykorzystywany tylko w tych miejscach, gdzie jest to konieczne z uwagi na zgodność z innymi systemami Unix, co wynika przede wszystkim z ograniczeń związanych ze stosowaniem tego algorytmu. Administratorzy systemu mają do wyboru szyfrowanie za pomocą wieloetapowego algorytmu DES, algorytmu MD5 lub algorytmu Blowfish. Wspomnieliśmy już, że jedną z zalet szyfrowania za pomocą algorytmu MD5 jest brak ograniczeń dotyczących długości hasła. Algorytm Blowfish (opracowany przez Bruce'a Shneiera i jego zespół) także szyfruje hasła dowolnej długości, ale także jest stosunkowo wolny. Z pozoru jest to sprzeczne z intuicją — wyjaśnimy to później, w podrozdziale „Kuba Rozpruwacz".

## Implementacja

System OpenBSD nie wykorzystuje architektury PAM, ale mimo to umożliwia skuteczne zarządzanie hasłami. Plik */etc/login.conf* zawiera dyrektywy dla algorytmów szyfrujących oraz definicje warunków, które muszą być spełnione przez użytkowników systemu. Wpisy w pliku *login.conf* zawierają więcej instrukcji związanych z wymaganiami stawianymi użytkownikom niż samych definicji strategii zarządzania hasłami. Omawiane w tym podrozdziale opcje powinny być dołączane na koniec opcji już istniejących. Pierwsza wartość każdego wpisu odpowiada typowi wyznaczonej dla użytkownika klasy logowania. Możemy użyć specjalnego wpisu z wartością `default` dla użytkowników bez zdefiniowanej klasy.

Aby uzyskać lub wyznaczyć klasę logowania dla danego użytkownika, należy otworzyć plik */etc/master.passwd* w narzędziu `vipw`. Klasa logowania znajduje się w piątym polu wpisu definiującego hasło użytkownika. Poniżej przedstawiamy przykład takich wpisów (klasy logowania oznaczyliśmy czcionką pogrubioną):

```
root:$2a$06$T22wQ2dH...:0:0:daemon:0:0:Fede:/root:/bin/csh
bisk:$2a$06$T22wQ2dH...:0:0:staff:0:0::/home/bisk:/bin/csh
```

Wpisy w pliku *login.conf* mogą mieć następującą postać (znak `\` reprezentuje w poniższym kodzie kontynuację w nowym wierszu):

```
default:\
:path=/usr/bin:\
        :umask=027:\
        :localcipher=blowfish,6
staff:\
        :path=/usr/sbin:\
        :umask=077:\
        :localcipher=blowfish,8
deamon:\
        :path=/usr/sbin:\
        :umask=077:\
        :localcipher=blowfish,8
```

Powyższy kod wymusza na systemie stosowanie algorytmu szyfrującego Blowfish dla każdego użytkownika. Użyte wartości `,6` i `,8` oznaczają liczby etapów przetwarzania haseł przez algorytm. Takie rozwiązanie oczywiście spowalnia działanie algorytmu, ponieważ zaszyfrowanie hasła wymaga znacznie więcej czasu. Jeśli jednak zaszyfrowanie hasła wymaga więcej czasu, także więcej czasu jest potrzebne na jego złamanie metodą testowania wszystkich możliwości. Przykładowo, łamanie hasła z wykorzystaniem słownika zawierającego 100 tysięcy słów potrwa znacznie dłużej, jeśli użyjemy 32-etapowego algorytmu szyfrującego (`localcipher=blowfish,32`) zamiast zastosowanego w tym przypadku algorytmu 6-etapowego.

Najważniejszymi wpisami w pliku *login.conf* są te rozpoczynające się od słowa `default` i `deamon`, ponieważ te pierwsze dotyczą wszystkich użytkowników, natomiast te drugie mają zastosowanie dla administratorów.

Każdy wpis może zawierać wiele opcji:

- `localcipher=`*`algorytm`* (wartość domyślna `old`) — definiuje wykorzystywany algorytm szyfrujący. Najlepszym rozwiązaniem jest zastosowanie wartości (algorytmów) `md5` i `blowfish,`*`n`*, gdzie *n* jest liczbą wykonywanych etapów szyfrowania (nie może przekraczać 32). Wartość `old` reprezentuje algorytm DES, którego stosowanie nie jest zalecane, ponieważ ogranicza długość haseł do ośmiu znaków, a wykorzystywane obecnie łamacze haseł działają przeciwko tak zaszyfrowanym hasłom bardzo efektywnie.
- `ypcipher=`*`algorytm`* — te same wartości, co w przypadku opcji `localcipher`. Opcja ta jest wykorzystywana wyłącznie w celu zapewnienia zgodności z rozproszonym logowaniem systemu NIS (Network Information System).
- `minpasswordlen=`*`n`* (domyślnie *n* = 6) — minimalna akceptowana długość hasła.
- `passwordcheck=`*`program`* — wskazuje na zewnętrzny program sprawdzający hasła. Opcja powinna być stosowana ostrożnie, ponieważ zewnętrzny program może być narażony na ataki koni trojańskich, może zawierać błędy i być podatny na atak polegający na przepełnieniu bufora.
- `passwordtries=`*`n`* (domyślnie *n* = 3) — liczba wyświetlanych przed użytkownikiem wezwań do podania nowego hasła w przypadku, gdy poprzednie hasło nie odpowiadało standardom systemu OpenBSD. Użytkownik nadal może ignorować te standardy, chyba że wartość tej opcji jest ustawiona na 0.

Zaktualizowany plik *login.conf* może mieć następującą postać (celowo definiujemy w poniższym przykładzie słabą klasę `ftpaccess`):

```
default:\
:path=/usr/bin:\
        :umask=027:\
        :localcipher=blowfish,8:\
        :minpasswordlen=8:\
        :passwordretries=0
ftpaccess:\
        :path=/ftp/bin:\
        :umask=777:\
        :localcipher=old:\
        :minpasswordlen=6:\
        :passwordretries=3
staff:\
        :path=/usr/sbin:\
        :umask=077:\
        :localcipher=blowfish,12:\
        :minpasswordlen=8:\
        :passwordretries=0
daemon:\
        :path=/usr/sbin:\
        :umask=077:\
        :localcipher=blowfish,31
```

Strategia zdefiniowana w powyższym pliku wymusza stosowanie algorytmu Blowfish dla haseł wszystkich użytkowników, z wyjątkiem tych należących do klasy `ftpaccess`. Strategia zarządzania hasłami dla tej klasy reprezentuje wymagania starych systemów Unix, o których wspomnieliśmy w opisie wartości `old` parametru `localcipher`. Hasła dla użytkowników klasy `staff` (klasy często wiązanej z uprawnieniami administracyjnymi) są szyfrowane w dwunastu etapach. Zgodnie z powyższymi zapisami, hasło administratora (użytkownika `root`), który domyślnie należy do klasy `deamon`, musi być szyfrowane przez wykonanie maksymalnej liczby etapów algorytmu Blowfish. Chociaż algorytmy Blowfish i MD5 obsługują dowolne długości haseł, w systemie OpenBSD wprowadzono ograniczenie do 128 znaków. Taka długość wystarczy nawet na krótki wiersz!


Jednymi z najlepszych miejsc do szukania haseł są pliki historii powłok użytkowników. Przejrzyj pliki *.history* i *.bash_history* w poszukiwaniu dziwnych poleceń. Zdarza się, że administrator przypadkowo wpisze hasło w wierszu poleceń. Takie sytuacje mają miejsce najczęściej wtedy, gdy administrator loguje się do zdalnego systemu lub używa polecenia `su` i popełnia błąd podczas wpisywania polecenia i odruchowo wyprzedza pojawienie się monitu o wprowadzenie hasła. Udało nam się kiedyś odkryć w ten sposób 13-znakowe hasło administratora!

# Kuba Rozpruwacz

Program John the Ripper (*http://www.openwall.com/john/*), czyli Kuba Rozpruwacz, jest prawdopodobnie najszybszym, najbardziej uniwersalnym i z pewnością jednym z najbardziej popularnych z dostępnych łamaczy haseł. Narzędzie obsługuje sześć różnych schematów kodowania haseł stosowanych w różnych odmianach systemów Unix oraz kodowanie Windows LANMan, znane także jako NTLM (używane w systemach

Windows NT, 2000 i XP). Program może stosować specjalizowane listy słów lub reguły tworzenia haseł oparte na typie i rozkładzie znaków. John the Ripper działa w co najmniej trzynastu różnych systemach operacyjnych i obsługuje wiele dostępnych obecnie procesorów, ze specjalnymi technikami optymalizacji procesorów Pentium i układów RISC włącznie.

## Implementacja

W pierwszej kolejności musimy pobrać i skompilować program. Najnowsza wersja programu została oznaczona kodem John-1.6.35, jednak będziemy musieli pobrać zarówno plik *John-1.6.35.tar.gz*, jak i starszy plik *John-1.6.tar.gz* (lub ich odpowiedniki z rozszerzeniem *.zip* dla systemów Windows). Wersja 1.6.35 nie zawiera wszystkich plików dokumentacji i wsparcia z oryginalnej wersji 1.6. Po rozpakowaniu archiwum John-1.6.35 w wybranym przez nas katalogu, musimy przejść do podkatalogu */src*:

```
[root@hedwig]# tar zxvf John-1.6.35.tar.gz
[root@hedwig]# tar zxvf John-1.6.tar.gz
[root@hedwig]# cd John-1.6.35
[root@hedwig John-1.6.35]# cd src
```

Następne polecenie jest proste: `make` *`nazwa systemu operacyjnego`*. Przykładowo, aby skompilować „Kubę" w środowisku Cygwin, musielibyśmy użyć polecenia `make win32-cygwin-x86-mmx`. W systemie BSD powinno wystarczyć polecenie `make freebsd-x86-mmx-elf`. Wpisanie samego polecenia `make` (bez argumentów) spowoduje wyświetlenie listy wszystkich obsługiwanych kombinacji systemów operacyjnych i procesorów.

```
[root@hedwig src]# make win32-cygwin-x86-mmx
```

Program zostanie następnie poddany automatycznej konfiguracji i kompilacji na naszej platformie. Kiedy proces ten się zakończy, wygenerowane pliki binarne i pliki konfiguracyjne zostaną umieszczone w katalogu *John-1.6.35/run*. Pobrana przez nas wersja nie zawiera kilku niezbędnych plików — będziemy je musieli samodzielnie wypakować z pliku *John-1.6.tar.gz* i umieścić w naszym podkatalogu */run*:

```
[root@hedwig]# cd John-1.6.35/run
[root@hedwig run]# cp ../../John-1.6/run/all.chr .
[root@hedwig run]# cp ../../John-1.6/run/alpha.chr .
[root@hedwig run]# cp ../../John-1.6/run/digits.chr .
[root@hedwig run]# cp ../../John-1.6/run/LANMan.chr .
[root@hedwig run]# cp ../../John-1.6/run/password.lst .
```

Jeśli wszystkie operacje kopiowania zakończą się pomyślnie, będziemy mogli od razu zająć się testowaniem programu. We wszystkich poniższych przykładach zakładamy, że znajdujemy się w katalogu *John-1.6.35/run*. Naszym pierwszym testem działania narzędzia będzie wygenerowanie danych odniesienia dotyczących szybkości łamania haseł w naszym systemie:

```
[root@hedwig run]# ./John -test
Benchmarking: Traditional DES [32/32 BS]... DONE
Many salts:     160307 c/s real, 161600 c/s virtual
Only one salt:  144627 c/s real, 146978 c/s virtual
```

```
Benchmarking: BSDI DES (x725) [32/32 BS]... DONE
Many salts:     5100 c/s real, 5280 c/s virtual
Only one salt:  5241 c/s real, 5262 c/s virtual

Benchmarking: FreeBSD MD5 [32/32 X2]... DONE
Raw:    3222 c/s real, 3248 c/s virtual

Benchmarking: OpenBSD Blowfish (x32) [32/32]... DONE
Raw:    234 c/s real, 239 c/s virtual

Benchmarking: Kerberos AFS DES [24/32 4K]... DONE
Short:  70348 c/s real, 72226 c/s virtual
Long:   192409 c/s real, 200426 c/s virtual

Benchmarking: NT LM DES [32/32 BS]...  DONE
Raw:    1807744 c/s real, 1875253 c/s virtual
```

Na naszą uwagę zasługują dwa testy wydajności: `FreeBSD MD5` oraz `NT LM DES`. Różnica pomiędzy wskaźnikami łamanych haseł na sekundę (`c/s`) tych dwóch testów jest ponad 500-krotna. Oznacza to, że pełny atak na zakodowane hasła polegający na sprawdzeniu wszystkich możliwości w przypadku systemu operacyjnego FreeBSD będzie trwał ponad 500 razy dłużej niż w przypadku systemu Windows NT! Zastosowanie w systemie FreeBSD algorytmu szyfrowania Blowfish dodatkowo wydłuży czas takiego ataku. Jest to doskonały przykład wpływu, jaki na łatwość złamania hasła tą metodą ma zastosowany algorytm szyfrujący. Zamiast mówić, że jeden algorytm jest bezpieczniejszy od drugiego, znacznie bardziej sprawiedliwe jest stwierdzenie, że algorytm Blowfish zapewnia mniejszą wrażliwość na atak metodą pełnego przeglądu.

## Łamanie haseł

Spróbujmy wreszcie złamać jakieś hasło. Program John the Ripper akceptuje trzy różne formaty plików z hasłami. W rzeczywistości jednak narzędzie to może złamać dowolne hasło zaszyfrowane w jednym z formatów wypisanych po wykonaniu polecenia `John` z opcją `-test`. Musimy jedynie umieścić to hasło w jednym z formatów akceptowanych przez aplikację. Jeśli używamy pliku *passwd* systemu Unix lub danych wyjściowych generowanych przez narzędzie pwdump (o którym wspomnimy w tym rozdziale nieco później), modyfikacje formatu pliku nie powinny być potrzebne. Poniżej przedstawiamy pięć różnych przykładów formatów plików z hasłami, które są prawidłowo interpretowane przez program John the Ripper (zakodowane hasła oznaczyliśmy pogrubioną czcionką):

1. `root:`**`rf5V5.Ce31sOE`**`:0:0::`
2. `root:`**`KbmTXiy.OxC.s`**`:11668:0:99999:7:-1:-1:1075919134`
3. `root:`**`$1$M9/GbWfv$sktn.4pPetd8zAwvhiB6.1`**`:11668:0:99999:7:-1:-1:1075919134`
4. `root:`**`$2a$06$v3LIuqqwOpX2M4iUnCVZcuyCTLX141yGNngtGSH4/dCqPHK8RyAie`**`:0:0::::::`
5. `Administrator:`**`500:66bf9d4b5a703a9baad3b435b51404ee`**`:17545362d694f996c37129225df11f4c:::`

Poniżej wymieniamy systemy, z których otrzymaliśmy pięć powyższych zakodowanych haseł. Zauważ, że chociaż istnieją zasadnicze różnice pomiędzy wymienionymi poniżej systemami operacyjnymi, odpowiednie formaty plików są podobne. Zwróć także uwagę na możliwość łamania haseł systemu Solaris za pomocą wersji programu John the Ripper

pracującej w systemie Windows — wymagana jest jedynie zakodowana wersja hasła, system operacyjny jest w tym przypadku nieistotny.

1. Solaris, algorytm DES, plik */etc/passwd*.
2. Mandrake Linux, algorytm DES, plik */etc/shadow*.
3. FreeBSD, algorytm MD5, plik */etc/shadow*.
4. OpenBSD, algorytm Blowfish, plik */etc/master.password*.
5. Windows 2000, program LAN Manager, plik *\WINNT\repair\SAM*.

Łamane hasła mogą być wykorzystywane w zabezpieczeniach aplikacji działającej w innym systemie niż Unix czy Windows. Aby złamać takie hasło, wystarczy skopiować jego zakodowaną wersję (w poniższych przykładach oznaczoną pogrubioną czcionką) do drugiego pola formatu pliku haseł systemu Unix:

- Urządzenia Cisco

  Oryginalny wpis: `enable secret 5 $1$M9/GbWfv$sktn.4pPetd8zAwvhiB6.1`

  Wpis dla programu John the Ripper: `cisco:$1$M9/GbWfv$sktn.4pPetd8zAwvhiB6.1::::`

- Pliki *.htaccess* serwera Apache, w których znajdują się kody haseł zaszyfrowane za pomocą algorytmu DES. Serwer Apache obsługuje także hasła zaszyfrowane za pomocą algorytmów SHA-1 i MD5, nie można ich jednak łamać za pomocą narzędzia John the Ripper.

  Oryginalny wpis w pliku *.htaccess*: `dragon:yJMVYngEA6t9c`

  Wpis dla programu John the Ripper: `dragon:yJMVYngEA6t9c::::`

- Inne hasła zaszyfrowane za pomocą algorytmu DES wykorzystywane przez takie aplikacje jak WWWBoard.

  Oryginalny wpis w pliku *passwd.txt*: `WebAdmin:aepTOqxOi4i8U`

  Wpis dla programu John the Ripper: `WebAdmin:aepTOqxOi4i8U:0:3:www.victim.com::`

Aby złamać plik z hasłami za pomocą domyślnych opcji programu John the Ripper, musimy podać nazwę tego pliku w wierszu polecenia. W przykładach prezentowanych w tym rozdziale będziemy używali trzech różnych plików z hasłami: pliku *passwd.unix* zawierającego hasła zaszyfrowane za pomocą algorytmu DES, pliku *passwd.md5* zawierającego hasła zaszyfrowane za pomocą algorytmu MD5 oraz pliku *passwd.LANMan* zawierającego hasła zaszyfrowane w systemie Windows NT:

```
[root@hedwig run]# ./John passwd.unix
Loaded 189 passwords with 182 different salts (Traditional DES [64/64 BS MMX])
```

Program John the Ripper automatycznie dobiera odpowiedni algorytm szyfrujący dla znalezionych w pliku zakodowanych haseł i rozpoczyna ich łamanie. Naciśnij dowolny klawisz, aby program wyświetlił na ekranie statystyki procesu łamania haseł; naciśnięcie kombinacji klawiszy *Ctrl+C* przerywa pracę aplikacji. Jeśli hasło zostanie złamane, John the Ripper wyświetla je na ekranie i zapisuje wraz z zaszyfrowanym kodem w celu późniejszego użycia. Aby przejrzeć wszystkie złamane hasła dla danego pliku, użyj opcji `-show`:

```
[root@hedwig run]# ./John -show passwd.unix
2buddha:smoothl:0:3:wwwboard:/:/sbin/sh
ecs:asdfgl:11262:0:40:5::11853:
informix:abc123:10864:0:40:5::12689:
kr:grant5:11569:0:35:5::11853:
mjs:rocky22:11569:0:35:5::11853:
np:nyObOy:11572:0:35:5::11853:
```

Wszystkie złamane hasła są zapisywane w pliku tekstowym *John.pot*, który rozrasta się wraz ze wzrostem liczby analizowanych przez nas haseł.

Słabe hasła, niezależnie od zastosowanego schematu szyfrowania, można złamać w czasie od kilku minut do jednego dnia. Łamanie lepszych haseł może wymagać tygodni lub nawet miesięcy, możemy jednak zastosować kilka sztuczek, które mogą nam pomóc w znacznie szybszym odgadywaniu haseł. Możemy używać skomplikowanych plików ze słownikami (z wyrazami obcymi, imionami, nazwami klubów sportowych, imionami bohaterów filmów science-fiction), możemy sprawdzać konkretne kombinacje haseł (zawierających przynajmniej dwie cyfry i znak interpunkcyjny) lub rozproszyć przetwarzanie na wiele komputerów.

Domyślnym słownikiem programu John the Ripper jest plik *password.lst*. Plik ten zawiera zbiór najczęściej spotykanych haseł. Wiele alternatywnych plików ze słownikami można bez trudu znaleźć w internecie — wystarczy wpisać odpowiednie zapytanie np. w wyszukiwarce *Google*. Jednym z najlepszych plików tego typu jest *bigdict.zip* (plik ma około 15 MB). Aby wymusić na programie użycie alternatywnego słownika, należy w wierszu polecenia wpisać opcję `-wordfile`:

```
[root@hedwig run]# ./John -wordflie:password.1st passwd.unix
Loaded 188 passwords with 182 different salts (Traditional DES [64/64 BS MMX])
guesses: 0  time: 0:00:00:01 100%  c/s: 333074  trying: tacobell - zhongguo
```

Za pomocą opcji `-rules` możemy nawet wykonywać pewne permutacje na słowach umieszczonych w stosowanym słowniku:

```
[root@hedwig run]# ./John -wordfile:password.1st -rules passwd.unix
Loaded 188 passwords with 182 different salts (Traditional DES [64/64 BS MMX])
guesses:  0 time: 0:00:00:58 100%  c/s: 327702  trying: Wonderin - Zenithin
```

Aby zrozumieć faktyczne działanie opcji `-rules`, zajrzyjmy do pliku konfiguracyjnego *John.conf* (lub pliku *John.ini*, w przypadku innych wersji programu). Poniżej przedstawiamy fragment tego pliku dotyczący właśnie permutacji stosowanych dla naszej listy słów (symbol # na początku wiersza oznacza komentarz):

```
[List.Rules:Wordlist]
# Try words as they are
:
# Lowercase every pure alphanumeric word
-c >3!?X1Q
# Capitalize every pure alphanumeric word
-c >2(?a!?XcQ
# Lowercase and pluralize pure alphabetic words
<*>2!?Alp
# Lowercase pure alphabetic words and append '1'
<*>2!?A1$1
```

Chociaż powyższe zapisy są z pozoru niemożliwe do rozszyfrowania, ich zrozumienie wcale nie jest takie trudne. Podstawowa składnia wielu z tych zasad pochodzi z narzędzia do łamania haseł napisanego przez Aleca Muffeta, libcrack. Wyobraźmy sobie, że strategia zarządzania hasłami w systemie wymaga, by każde hasło rozpoczynało się od cyfry. W takim przypadku jest oczywiste, że nie będziemy testowali hasła „letmein", ponieważ nie spełnia ono reguł narzuconych przez strategię, poprawne może być np. hasło „7letmein". Oto reguła dla haseł rozpoczynających się od cyfry:

```
# Dodaje cyfrę na początku każdego hasła (powoduje wykonanie 10 razy więcej
# iteracji przez listę wyrazów)
^[0123456789]
```

Możemy rozłożyć powyższą regułę na trzy części. Symbol ^ oznacza, że operacja powinna być wykonywana na początku słowa. Innymi słowy, następujący po tym symbolu znak powinien zostać dołączony na początek każdego słowa. Nawiasy kwadratowe `[` i `]` definiują cały zbiór znaków, zamiast pojedynczego znaku po symbolu ^. Cyfry `0123456789` są konkretnymi znakami dodawanymi na początku każdego słowa. Jeśli więc reguła zostanie zastosowana dla słowa „letmein", wygenerowanych zostanie dziesięć testowanych słów, od „0letmein" do „9letmein".

Poniżej przedstawiamy reguły definiujące miejsca wstawiania znaków:

| Symbol | Opis | Przykład |
|---|---|---|
| ^ | Na początku wyrazu | `^[01]`<br>`0letmein`<br>`1letmein` |
| $ | Na końcu wyrazu | `$[!.]`<br>`letmein!`<br>`letmein.` |
| i[*n*] | Na *n*-tej pozycji w wyrazie | `i[4][XZ]`<br>`letXmein`<br>`letZmein` |

Możemy określić dowolny zakres wstawianych znaków. Dla każdego dodatkowego znaku ponownie przeglądana jest cała lista. Przykładowo, jeśli do każdego słowa dodamy na początek jedną z dziesięciu cyfr (0 – 9), lista tysiąca słów rozszerzy się w efekcie do listy zawierającej dziesięć tysięcy słów. Oto kilka innych przydatnych znaków, które możemy dodawać do wyrazów z listy:

- `[0123456789]` — cyfry
- `[!@#$%^&*0]` — cyfry z przytrzymanym klawiszem *Shift*
- `[,.?!]` — znaki interpunkcyjne

Możemy zastosować reguły konwersji do zmiany wielkości lub typu (wielkie litery, małe litery, zamiana litery e na cyfrę 3) znaków lub do usunięcia pewnych typów znaków:

- `?v` — klasa samogłosek (*a*, *e*, *i*, *o*, *u*)
- `s?v.` — zastępuje samogłoski znakiem kropki

- `@@?v` — usuwa wszystkie samogłoski
- `@@a` — usuwa wszystkie litery *a*
- `sa4` — zastępuje wszystkie litery *a* cyfrą 4
- `se3` — zastępuje wszystkie litery *e* cyfrą 3
- `l*` — gwiazdka reprezentuje literę zmienianą na małą
- `u*` — gwiazdka reprezentuje literę zmienianą na wielką

Reguły są doskonałym sposobem na poprawienie wskaźnika trafień w procesie odgadywania haseł, szczególnie przydatne okazują się reguły dołączające znaki i zamieniające litery na cyfry. Reguły były jeszcze bardziej korzystne w czasach, gdy wydajność procesorów nie była dużo większa od małpy próbującej użyć liczydła. Obecnie, kiedy za kilkaset dolarów można kupić komputer z procesorem o częstotliwości taktowania przekraczającej 2 GHz, nie ma większego znaczenia, czy zastosujemy skomplikowane reguły, czy od razu przystąpimy do ataku polegającego na przetestowaniu wszystkich możliwości.

Skomplikowane reguły i rozbudowane słowniki nie dają oczywiście gwarancji złamania każdego hasła. Doszliśmy w ten sposób do zagadnień związanych z atakiem polegającym na przetestowaniu wszystkich możliwości. Innymi słowy, będziemy sprawdzali każdą kombinację znaków dla określonej długości słowa. Program John the Ripper domyślnie przełącza się w tryb takiego ataku, jeśli w wierszu polecenia nie przekażemy żadnych opcji. Aby wymusić na programie stosowanie konkretnej metody ataku, używamy opcji `-incremental`:

```
[root@hedwig run]# ./John -incremental:LANMan passwd.LANMan
Loaded 1152 passwords with no different salts (NT LM DES [64/64 BS MMX])
```

Domyślny plik *John.conf* zawiera cztery różne opcje:

- `All` — małe litery, wielkie litery, cyfry, znaki interpunkcyjne, znaki z wciśniętym klawiszem *Shift*
- `Alpha` — małe litery
- `Digits` od 0 do 9
- `LANMan` — podobnie jak w przypadku `All`, ale bez małych liter

Dla każdej z tych opcji definiuje się w pliku *John.conf* pięć pól. Przykładowo, wpis `LANMan` zawiera następujący zestaw pięciu pól i wartości:

- `[Incremental:LANMan]` — opis opcji
- `File = ./LANMan.chr` — wykorzystywany plik z listą znaków
- `MinLen = 0` — minimalna długość generowanego ciągu znaków
- `MaxLen = 7` — maksymalna długość generowanego ciągu znaków
- `CharCount = 69` — liczba znaków na liście

A takie wartości są domyślnie zdefiniowane dla wpisu `All`:

- `[Incremental:All]` — opis opcji
- `File = ./all.chr` — wykorzystywany plik z listą znaków
- `MinLen = 0` — minimalna długość generowanego ciągu znaków
- `MaxLen = 8` — maksymalna długość generowanego ciągu znaków
- `CharCount = 95` — liczba znaków na liście

Pola `MinLen` i `MaxLen` są najważniejsze, ponieważ od ich wartości zależy skuteczność i czas trwania ataku. Wartość pola MaxLen w przypadku zakodowanych haseł LANMan nigdy nie będzie większa od siedmiu znaków. Jeśli podniesiemy wartość pola `CharCount` do potęgi `MaxLen`, uzyskamy liczbę kombinacji wykorzystywanych podczas pełnego ataku polegającego na przetestowaniu wszystkich możliwości. Przykładowo, łączna liczba kombinacji dla haseł LANMan wynosi około 7,6 tryliona, a łączna liczba kombinacji generowanych w trybie `All` wynosi aż 6700 trylionów! Zauważ, że stosowanie trybu `incremental:All` jest bezcelowe w przypadku haseł zakodowanych metodą LANMan, ponieważ weryfikowanie małych i wielkich liter jest w tym przypadku zbędne.

Jeśli dysponujemy listą haseł systemu Unix, o którym wiemy, że wszystkie stosowane hasła składają się z dokładnie ośmiu znaków, powinniśmy zmodyfikować odpowiednią opcję. W tym przypadku byłoby stratą czasu wykorzystywanie programu John the Ripper do odgadywania haseł zawierających siedem i mniej znaków:

```
[Incremental:All]
File = ./all.chr
MinLen = 8
MaxLen = 8
CharCount = 95
```

Możemy teraz uruchomić nasze narzędzie:

```
[root@hedwig run]# ./John -incremental:All passwd.unix
```

Wygenerowane zostaną tylko ośmioznakowe ciągi. Aby to sprawdzić, możemy dodatkowo użyć opcji `-stdout`. Spowodujemy w ten sposób wyświetlenie na ekranie każdego wygenerowanego słowa:

```
[root@hedwig run]# ./John -incremental:All -stdout
```

Takie rozwiązanie może być korzystne, jeśli chcemy przekierować standardowe wyjście do pliku i stworzyć tym samym ogromną listę słów z myślą o przyszłych zastosowaniach w programie John the Ripper lub dowolnym innym narzędziu umożliwiającym wykorzystywanie pliku z listą wyrazów, np. programie Nessus lub THC-Hydra.

```
[root@hedwig run]# ./John -makechars:guessed
Loaded 3820 plaintexts
Generating charsets... 1 2 3 4 5 6 7 8 DONE
Generating cracking order... DONE
Successfully written charset file: guessed (82 characters)
```

## Pliki odtwarzania i rozproszone łamanie haseł

Aby sprawnie zarządzać ogromnymi zbiorami haseł na różnych etapach procesu ich łamania, musisz zrozumieć kilka istotnych zasad dotyczących funkcjonowania programu John the Ripper. Narzędzie to okresowo zapisuje swój stan w specjalnym pliku odtwarzania. Częstotliwość takich zapisów można ustawić w pliku konfiguracyjnym *John.conf*:

```
# Crash recovery file saving delay in seconds
Save = 600
```

Domyślną nazwą pliku odtwarzania jest *restore*, można ją jednak zmienić za pomocą opcji `-session`.

```
[root@hedwig run]# ./John -incremental:LANMan -session:pdc \
> passwd.LANMan
Loaded 1152 passwords with no different salts (NT LM DES
[64/64 BS MMX])
```

Zawartość pliku odtwarzania będzie podobna do poniższej:

```
REC2
5
-incremental:LANMan
-session:pdc
passwd.LANMan
-format:lm
6
0
47508000
00000000
0
-1
488
0
8
3
2
6
5
2
0
0
0
```

Dziewiąty i dziesiąty wiersz w powyższym pliku (oznaczone pogrubioną czcionka) zawierają szesnastkową wartość reprezentującą łączną liczbę wygenerowanych i sprawdzonych wyrazów. Liczba możliwych kombinacji znacznie przekracza 32-bitową wartość, którą można w ten sposób reprezentować, zatem program John the Ripper wykorzystuje dwa 32-bitowe pola do stworzenia liczby 64-bitowej. Znajomość tych wartości i umiejętność ich interpretowania i modyfikowania jest przydatna podczas wykonywania rozproszonego łamania haseł. Przykładowo, weźmy nasz plik przywracania i spróbujmy go wykorzystać do zainicjowania na dwóch osobnych komputerach dwóch równoczesnych ataków polegających na testowaniu wszystkich możliwości. Plik przywracania dla pierwszego komputera miałby następującą zawartość:

```
REC2
4
-incremental:LANMan
passwd.LANMan
-format:lm
4
0
00000000
00000000
0
-1
333
0
8
15
16
0
0
0
0
0
0
```

Natomiast plik przywracania dla drugiego komputera miałby następującą zawartość:

```
REC2
4
-incremental:LANMan
passwd.LANMan
-format:lm
4
0
00000000
0000036f
0
-1
333
0
8
15
16
0
0
0
0
0
0
```

Pierwszy system rozpocznie atak od pierwszej kombinacji (od zerowego stanu licznika). Drugi komputer rozpocznie testowanie kodów LANMan od wartości licznika `0000036f 00000000`. Praca została teraz rozdzielona pomiędzy dwa komputery w taki sposób, że nie musimy się obawiać powtarzania tych samych kombinacji. Dobrą techniką znajdowania właściwej wartości licznika jest uruchomienie systemu na określony czas.

Przykładowo, wyobraźmy sobie, że dysponujemy skromnym zbiorem dziesięciu komputerów. W każdym z tych systemów program John the Ripper może testować około

400 000 haseł w ciągu sekundy. Przetestowanie wszystkich siedmioznakowych kombinacji dla typowego kodu LANMan ($69^7$ kombinacji) zajęłoby jednemu komputerowi około trzydziestu tygodni. Uruchamiamy więc program w jednym z tych systemów na tydzień. Po upływie tygodnia zapisujemy wartość licznika. Wykorzystujemy ją jako wartość początkową w pliku odtwarzania w drugim systemie, mnożymy ją przez dwa i używamy jako wartości początkowej w pliku odtwarzania w kolejnym systemie. Teraz nasze 10 systemów może przeprowadzić pełny atak tylko w trzy tygodnie. Poniżej przedstawiamy prościutki algorytm obliczania mnożnika dla liczników, *X*, który będzie nam potrzebny do stworzenia dziesięciu plików odtwarzania — po jednym dla każdego systemu. Pierwszy system będzie rozpoczynał swoją pracę od zerowej wartości licznika, kolejny rozpocznie proces przetwarzania od wartości licznika pierwszego systemu powiększonej o *X*, itd.:

Łączna liczba tygodni:

$Tw = (69^7/\text{hasła na sekundę})/(\text{hasła na tydzień})$
$Tw = (69^7/400\,000)/(604\,800) = 30{,}8$ tygodni

Mnożnik dla licznika:

$X = Tw/(10 \text{ systemów})$
$X = 30{,}8/10 = 3$

Wartość licznika po tygodniu (liczba szesnastkowa, skopiowana z pliku odtwarzania):

00030000 00000000

Poniżej obliczamy wartości liczników dla systemów rozproszonych (w notacji szesnastkowej). Wartości te musimy koniecznie umieścić w pliku odtwarzania każdego z systemów:

System 1. = 0
System 2. = *licznik***X* = 00090000 00000000
System 3. = *licznik***X**2 = 00120000 00000000
System *n*-ty = *licznik***X**(*n*–1) = wartość w pliku odtwarzania
System 10. = *licznik***X**9 = 00510000 00000000

Zaproponowana metoda nie ma nic wspólnego z eleganckim algorytmem, ale sprawdza się w przypadku kilku identycznych komputerów. Inna metoda rozpraszania przetwarzania polega na zastosowaniu opcji `-external`. Mówiąc najprościej, opcja ta umożliwia nam pisanie własnych plików z definicjami stworzonych przez nas funkcji odgadywania haseł. Zewnętrzne procedury są przechowywane w pliku *John.conf* w sekcji dyrektyw `List.External`. Wystarczy więc w wierszu poleceń wpisać opcję `-external` z interesującą nas dyrektywą:

```
[root@hedwig run]# ./John -external:Parallel passwd.LANMan
```


Jeśli zamierzasz wykorzystać tę metodę, upewnij się, że zmieniłeś wiersz `node=1` na `node=2` w pliku *John.conf* na drugim komputerze. Pamiętaj także, że implementacja tej metody nie jest efektywna dla więcej niż dwóch komputerów, ponieważ instrukcja `if (number++ % total)` spowoduje w niektórych systemach testowanie powtarzających się słów.

## Czy to działa także w moim systemie?

Najlepszą oznaką działania programu John the Ripper w naszym systemie jest stałe obciążenie procesora. Możemy przeanalizować listę procesów (za pomocą polecenia `ps` w systemach Unix lub na liście procesów w Menedżerze zadań systemów Windows), jednak nie znajdziemy tam informacji o procesie John. Jeśli spróbujemy zmienić nazwę pliku wykonywalnego, np. na „*inetd* ” (zwróć uwagę na *celowo użytą spację na końcu nazwy*), program nie będzie działał bez dodatkowego zmodyfikowania kilku wierszy kodu źródłowego.

**Studium przypadku: strategie atakowania haseł**

Reguły definiowane w pliku *John.conf* pozwalają na znaczną modyfikację stosowanego słownika. Wspomnieliśmy już o prostej regule dodającej cyfrę na początku każdego odgadywanego hasła:

```
# Dodaje cyfrę na początku każdego hasła (powoduje wykonanie 10 razy więcej
# iteracji przez listę wyrazów)
^[0123456789]
```

Jakie inne scenariusze są możliwe? Co możemy zrobić, jeśli zauważyliśmy tendencję w schemacie haseł administratora stosowanym w konkretnych sieciowych systemach Unix? Przykładowo, załóżmy, że chcemy stworzyć listę słów złożoną z wszystkich kombinacji wielkich i małych liter słowa *bank*. Odpowiednia reguła w pliku *John.conf* miałaby następującą postać:

```
# Permutacja słowa "ban" (łącznie 8 haseł)
i[0][bB]i[1][aA]i[2][nN]
```

Nietrudno zauważyć, że w powyższej regule użyliśmy tylko pierwszych trzech liter interesującego nas wyrazu. Wynika to z faktu, że program John the Ripper musi dysponować listą słów, na której będzie działał. W tym przypadku plik z taką listą, nazwany *password.lst*, powinien zawierać dwie litery:

```
k
K
```

Jeśli uruchomimy teraz nasze narzędzie z nową regułą i skróconym plikiem *password.lst*, otrzymamy następujący wynik:

```
$ ./John.exe -wordfile:password.lst -rules -stdout
bank
bank
bank
bank
bank
bank
bank
bank
Bank
BanK
BaNk
BaNK
Bank
BAnK
BANk
BANK
words:  16  time: 0:00:00:00 100%  w/s: 47.05  current: BANK
```

A oto inna reguła zdefiniowana z myślą o złamaniu hasła, które stworzono zgodnie ze strategią zarządzania hasłami wymagającą, by na trzeciej pozycji znajdował się znak specjalny, oraz by na ostatniej pozycji znajdowała się cyfra:

```
# Dokładna strategia (powoduje wykonanie 160 razy więcej
# iteracji przez listę wyrazów)
i[2][`~!@#$%^&*()-_=+]$[0123456789]
```

Poniżej przedstawiamy skrócony przykład wygenerowanych danych wyjściowych dla testowanego wyrazu *password*:

```
$ ./John.exe -wordfile:password.lst -rules -stdout
pa`ssword0
pa`ssword1
pa`ssword2
...
pa~ssword7
pa~ssword8
pa~ssword9
pa!ssword0
pa!ssword1
pa!ssword2
...
```

Jak widać, możemy bez trudu stworzyć reguły odpowiadające opracowanym przez administratorów zasadom konstruowania haseł w komputerach podłączonych do sieci.

# L0phtCrack

Z początku wydawało się, że systemy Windows oferują udoskonalone mechanizmy zabezpieczania komputerów hasłami w porównaniu z wydawanymi w tym samym czasie systemami Unix. W większości starych systemów Unix nie można było tworzyć haseł dłuższych niż ośmioznakowe.

Autorzy systemu Windows NT szczycili się maksymalną długością czternastu znaków, czyli niemal dwukrotnie większą niż w oferowanych w tamtym czasie w systemach Unix! Wówczas jednak Mudge i Weld Pond z firmy L0pht Heavy Industries zdemaskowali słabość szyfrowania LANMan. Ich firma szybko wypuściła na rynek narzędzie wykorzystujące niedociągnięcia w schemacie szyfrowania haseł firmy Microsoft.

Wspominaliśmy już w tym rozdziale kilkakrotnie o hasłach zakodowanych zgodnie ze schematem szyfrowania LANMan. Wiemy, że zakodowana reprezentacja hasła użytkownika przypomina wpisy w plikach /etc/passwd lub /etc/shadow w systemach Unix. Naszym celem jest w tym momencie przeanalizowanie sposobu generowania i przechowywania tych reprezentacji. System Windows przechowuje dwie wersje każdego hasła użytkownika. Pierwsza wersja nosi nazwę kodu LANMan lub kodu LM. Druga jest zwykle nazywana kodem NT — hasło jest szyfrowane za pomocą jednokierunkowej funkcji MD4, co oznacza, że hasło można zaszyfrować, ale nigdy nie można go odszyfrować. Także kod LANMan jest generowany na podstawie funkcji jednokierunkowej, jednak w tym przypadku hasło przed szyfrowaniem (za pomocą algorytmu DES) jest dzielone na dwie połowy.

Spójrzmy na moment na zawartość trzech kodów LANMan dla trzech różnych haseł. Każde z nich jest reprezentowane w notacji szesnastkowej i składa się z szesnastu bajtów:

```
898f30164a203ca0 14cc8d7feb12cldb
898f30164a203ca0 aad3b435b51404ee
14cc8d7feb12cldb aad3b435b51404ee
```

Nietrudno zauważyć podobieństwa pomiędzy tymi trzema przykładami. Ostatnie osiem bajtów drugiego i trzeciego przykładu jest identyczne: `aad3b435b51404ee`. Taka sama wartość pojawi się w drugiej połowie każdego kodu wygenerowanego na podstawie hasła mającego mniej niż osiem znaków. Jest to niewątpliwe niedociągnięcie autorów schematu szyfrującego z dwóch powodów: zaszyfrowana postać hasła wskazuje na to, że nie jest ono dłuższe niż siedem znaków oraz ujawnia brak jakiegokolwiek związku funkcji generującej drugą połowę kodu z informacjami zawartymi w pierwszej połowie. Zwróć uwagą na fakt, że druga połowa w pierwszym przykładzie (`14cc8d7feb12cldb`) jest identyczna z pierwszą połową w trzecim przykładzie. Oznacza to, że hasło jest szyfrowane na podstawie dwóch niezależnych siedmioznakowych zbiorów, zamiast wprowadzenia zależności pomiędzy obiema częściami generowanego kodu.

W efekcie potencjalne czternastoznakowe hasło użytkownika jest zamieniane na dwa mniejsze, siedmioznakowe hasła. Co więcej, schemat szyfrowania LANMan ignoruje wielkość liter, co znacznie (około dziesięć razy) skraca czas potrzebny na wykonanie pełnego ataku metodą testowania wszystkich możliwości.

## Implementacja

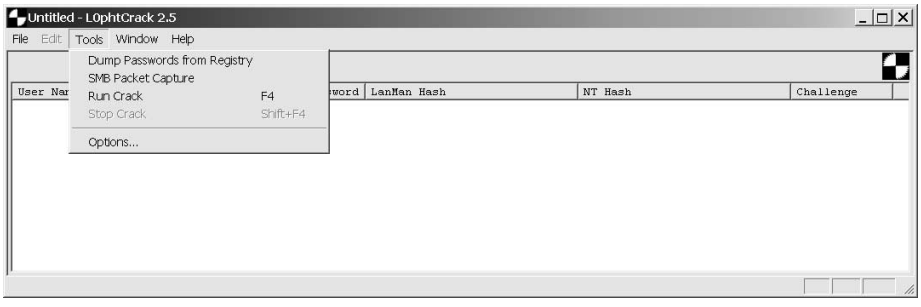
Program L0pthCrack wprowadził łamanie haseł do bogatego graficznego interfejsu użytkownika systemów Windows NT i jego następców, Windows 2000 i XP. Próba zdobycia haseł przechowywanych w systemach Unix wymaga zwykle przejęcia pliku */etc/passwd* lub */etc/shadow* — oba mają łatwy do odczytania format tekstowy. Systemy Windows przechowują hasła w pliku binarnym Security Accounts Manager (SAM), którego interpretacja bez specjalnych narzędzi jest dosyć trudna. Narzędzie L0pthCrack nie tylko oferuje funkcje odgadywania haseł, umożliwia także wydobywanie zaszyfrowanych kodów LANMan z dowolnego pliku SAM czy lokalnego lub zdalnego systemu, a nawet przechwytywanie takich kodów przesyłanych w sieci.

Plik SAM jest przechowywany w katalogu *\WINNT\system32\config*. Jeśli spróbujemy ten plik skopiować lub otworzyć, otrzymamy komunikat o błędzie:

```
C:\WINNT\system32\config\copy SAM c:\temp
The process cannot access the file because it is being used by
another process.
        0 file(s)  copied.
```

Nie poddawaj się! Pomocna okazuje się tworzona przez system Windows kopia zapasowa pliku SAM w katalogu *\WINNT\repair* (niekiedy umieszczana w katalogu *\WINNT\repair\RegBack*).

Program L0pthCrack wydobywa hasła z lokalnych lub zdalnych komputerów po wybraniu z menu *Tools* opcji *Dump Passwords From Registry*.

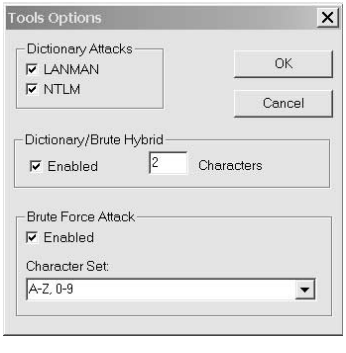



Do zdalnego wydobywania haseł niezbędna jest poprawna sesja połączenia z zasobem ADMIN$. To zaś wymaga dostępu do portu 139. protokołu NetBIOS TCP. Do ustanowienia takiej sesji możemy wykorzystać program L0pthCrack, możemy także zrobić to samodzielnie:

```
C:\>net use \\victim\admin$ * /u:Administrator
Type the password for \\localhost\admin$:
The command completed successfully.
```

Program może także przechwytywać kody LANMan przesyłane w sieci. Za każdym razem, gdy narzędzie `net use` otrzymuje dostęp do zdalnego komputera, wydobywany jest odpowiedni zaszyfrowany kod uwierzytelniający. Takie rozwiązanie wymaga jednak dostępu do lokalnej sieci i możliwości analizy występującej w niej komunikacji sieciowej, zatem możliwości stosowania tego mechanizmu są dosyć ograniczone.

Szybkość łamania haseł w programie L0pthCrack może budzić podziw, jednak tylko do czasu, kiedy porównamy wydajność tego narzędzia z najnowszymi wersjami aplikacji John the Ripper. L0pthCrack nie oferuje jednak wyłącznie wszechstronności w zakresie modyfikowania reguł, w oknie Options możemy także dostosować wykorzystywaną listę znaków.




Często najlepszym rozwiązaniem jest jednak zastosowanie programu L0pthCrack do wydobycia haseł, zapisania ich w pliku haseł (menu *File*, opcja *Save As*) dla narzędzia John the Ripper.

Aby program John the Ripper mógł przetwarzać tak utworzony plik, będziemy go musieli zmodyfikować. Konieczne zmiany polegają na umieszczeniu uzyskanych kodów haseł w odpowiednich polach.

Oto przykładowy plik wygenerowany przez program L0pthCrack:

```
LastBruteIteration=0
CharacterSet=1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ
ElapsedTime=0 0
Administrator:"":"": A34E6990556D7BA3BA1F6705936BF461:
2B1437DBB1DC57DA3DA1B88BADAB13B2:::
```

Poniżej przedstawiono natomiast plik dla narzędzia John the Ripper. Zauważ, że pierwsze trzy wiersze oryginalnego pliku zostały usunięte, oraz że istnieje tylko jedno pole pomiędzy nazwą użytkownika (`Administrator`) a zaszyfrowanym hasłem. Zawartość tego pola nie ma żadnego znaczenia dla przebiegu łamania hasła w programie John the Ripper, umieściliśmy w tym miejscu systemowy identyfikator użytkownika (SID) jedynie dla przypomnienia:

```
Administrator:500:A34E6990556D73A3BA1F6705936BF461:2B1437DBB1DC57DA3DA1B88BADAB13B2:::
```

W wersji 3.0 programu L0pthCrack wprowadzono udoskonalony mechanizm testowania możliwości aplikacji. Chociaż nowa wersja oferuje wygodniejsze funkcje przydatne szczególnie administratorom systemów (np. opcja informująca jedynie o zakończonej sukcesem próbie złamania hasła, ale nie wyświetlająca otrzymanego wyniku), najlepszym rozwiązaniem jest w naszym przypadku stosowanie programu L0phtCrack 2.52 do zdobywania haseł oraz wykorzystywanie narzędzia John the Ripper do ich łamania.

Używanie wersji 3.0 programu L0phtCrack ma oczywiście swoje zalety. W domenach systemu Windows 2000 mogą istnieć konta z 15-znakowymi hasłami. To wyklucza możliwość ich szyfrowania zgodnie ze schematem LANMan (który ogranicza długość haseł do 14 znaków). W takim przypadku dla 15-znakowych haseł wersja 2.5 wyświetli komunikat o braku hasła zarówno dla kodu LANMan, jak i kodu NTLM. Takie konta prawidłowo wczyta i zidentyfikuje natomiast wersja 3.0. Jeśli kiedykolwiek zetkniesz się z kodem podobnym do poniższego:

```
AAD3B435B51404EEAAD3B435B51404EE:FA95F45CC70B670BD865F3748CA3E9FC:::
```

będzie to oznaczało, że masz do czynienia z jednym z takich „superhaseł". Zauważ, że powyższy kod zawiera w części LANMan (oznaczonej czcionką pogrubioną) dwukrotnie powtórzony kod `AAD3B435B51404EE`, o którym wiemy, że reprezentuje puste hasło.

Inną zaletą wersji 3.0 programu L0phtCrack jest możliwość rozpraszania procesu łamania haseł. Metoda rozproszonego łamania polega na podzieleniu całego procesu na odpowiednie bloki. Jest to istotna zaleta tego narzędzia, szczególnie w środowisku systemów heterogenicznych, w których możemy efektywnie przeprowadzać atak i stale śledzić postęp naszych działań.

## Ochrona naszych haseł

Silne zabezpieczenia całej sieci i należących do niej komputerów są najlepszym sposobem ochrony plików z hasłami i samych haseł. Jeśli użytkownik może skopiować plik z hasłami lub przynajmniej ich zaszyfrowane kody do systemu Windows, złamanie większości haseł jest kwestią nieodległej przyszłości. Pamiętaj jednak, że takie narzędzia jak John the Ripper czy L0phtCrack nie obsługują pewnych znaków akceptowanych w systemach Windows jako prawidłowe.

Wiele kombinacji klawisza *Alt* i klawiszy klawiatury numerycznej zwraca znaki, które nie są prawidłowo rozpoznawane przez dostępne obecnie łamacze haseł. Aby wprowadzić do hasła jedną z takich kombinacji musisz koniecznie wykorzystywać cyfry z klawiatury numerycznej. Przykładowo, zestaw liter *p-a-s-s-w-Alt-242-r-d* (czyli *passwòrd*) pozostanie bezpieczny do czasu, gdy ktoś odpowiednio zaktualizuje swoje narzędzia łamiące. Dodanie znaków dostępnych po zastosowaniu kombinacji *Alt-nnn* znacznie wydłuża atak polegający na przetestowaniu wszystkich możliwości.


Kombinacje z klawiszem *Alt* dla znaków specjalnych rozpoczynają się od liczby *160* (kombinacja *Alt+160*) i kończą na liczbie *255*.

## Usuwanie kodów LANMan

Z punktu widzenia administratorów dbających o bezpieczeństwo zarządzanych przez siebie systemów, niewątpliwą zaletą systemów Windows XP i Windows 2000 Service Pack 2 jest klucz Rejestru umożliwiający usunięcie przechowywanych kluczy LANMan dla haseł użytkownika. Pamiętajmy, że kod LM jest najsłabszą wersją zaszyfrowanego hasła użytkownika, ponieważ ignoruje różnice pomiędzy wielkimi i małymi literami. Bez ograniczeń wynikających ze stosowania schematu LANMan (patrz omówienie implementacji programu L0phtCrack), możemy tworzyć hasła 15-znakowe lub dłuższe. Możemy także tak ustawić poniższy klucz, by zakazywał systemowi Windows przechowywanie kodów LANMan dla wszystkich haseł zmienianych w przyszłości:

```
HKLM\SYSTEM\CurrentControlSet\Control\Lsa\NoLMHash
```

Wartość `NoLMHash` typu *REG_DWORD* powinna być równa 1. Takie rozwiązanie spowoduje niezgodność naszego systemu z wszystkimi systemami Windows z serii *9x* i Me, ale w żaden sposób nie wpłynie na zgodność z systemami 2000 i XP. Kiedy już ustawisz tę wartość, upewnij się, że wszyscy użytkownicy zmienili swoje hasła — tylko w ten sposób możemy wymusić obowiązywanie nowych ustawień. Jeśli nadal sądzisz, że ta prosta zmiana wartości Rejestru nie wpływa na poprawę bezpieczeństwa systemu, przeanalizuj następujący przykład. Dla ośmioznakowego hasła różnica w przestrzeni przeszukiwania (która jest równoznaczna z różnicą czasu potrzebnego do przeprowadzenia pełnego ataku polegającego na przetestowaniu wszystkich możliwości) pomiędzy kodem LANMan a kodem MD4 jest ponad tysiąckrotna! Innymi słowy, istnieje $69^7$ kombinacji dla kodu LANMan (pamiętaj, że rozważamy ośmioznakowe hasło, które w tym schemacie szyfrowania jest rozkładane na hasło siedmioznakowe i hasło jednoznakowe) oraz $96^8$ kombinacji dla kodu MD4.

**Studium przypadku: wykrywanie działania programu L0phtCrack w naszym systemie**

Oprogramowanie antywirusowe może wskazać na program L0phtCrack jako program stwarzający zagrożenie. Wynika to z faktu, że jest to bardzo przydatne narzędzie kontrolne w rękach administratorów, ale także bardzo skuteczna broń w rękach hakerów, którzy instalują tę aplikację bez odpowiednich uprawnień. Program L0phtCrack pozostawia ślady w postaci plików z rozszerzeniem *.lc*. Jeśli narzędzie to jest aktualnie zainstalowane w systemie (nie jest uruchamiane z dyskietki), możemy przeanalizować Rejestr w poszukiwaniu ciągu znaków *l0pht*. Przyjrzyjmy się teraz kilku testom, które może wykonać administrator systemu po dokonaniu odkrycia wskazującego na dostęp stacji roboczej jednego z pracowników do zasobu ADMIN$ kontrolera PDC.

Pominiemy kilka mniej istotnych kroków, np. sprawdzanie danych na stacji roboczej i analiza uruchamianych poleceń. Zamiast tego skupimy się wyłącznie na hasłach w naszej sieci. Pracując w firmie zatrudniającej ponad 600 osób, musielibyśmy poświęcić mnóstwo czasu na dokładne sprawdzenie hasła każdego z pracowników; jeśli jednak od razu spróbujemy znaleźć bezpośrednie dowody łamania haseł na komputerze podejrzanego użytkownika, będziemy musieli przeanalizować tylko niewielką ilość kluczowych danych. Najbardziej oczywistym dowodem przeprowadzonej na badanym komputerze instalacji programu L0phtCrack jest generowany przez tę aplikację klucz Rejestru:

```
HKLM\SYSTEM\Software\L0pht Heavy Industries\L0phtcrack 2.5
```

Niestety, w podejrzanym systemie nie znaleźliśmy takiego klucza. Istnieją jednak inne ślady instalacji programu w podejrzanym systemie. Jeden z kluczy odwołuje się do sterownika przechwytywania pakietów, który jest wykorzystywany przez program L0phtCrack do nasłuchiwania sieci i wyłapywania przesyłanych w niej kodów LANMan:

```
HKLM\SYSTEM\CurrentControlSet\Service\NDIS3Pkt
```

Klucz ten mógł oczywiście zostać utworzony przez inne programy, jednak ustawiana przez nie poprawna wartość powinna mieć następującą postać (zwróć uwagę na wielkość liter):

```
HKLM\SYSTEM\CurrentControlSet\Service\Ndis3pkt
```

W badanym Rejestrze znaleźliśmy klucz *NDIS3Pkt*, możemy więc podejrzewać, że w systemie został zainstalowany program L0phtCrack. Sprytny pracownik mógł próbować zlikwidować większość śladów obecności tego narzędzia; jeśli był bardzo ostrożny, mógł nawet defragmentować i nadpisać wolną przestrzeń dysku twardego celem zabezpieczenia się przed zaawansowanymi narzędziami odnajdującymi dane usunięte z dysku. System Windows przechowuje jednak jeszcze jeden wpis zawierający informacje o deinstalacji aplikacji L0phtCrack. Nawet po prawidłowym usunięciu programu, w Rejestrze jest pozostawiany następujący klucz:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\L0phtcrack 2.5
```

Jeśli administrator systemu znajdzie taki wpis w Rejestrze, może być absolutnie pewny, że w badanym systemie zainstalowano kiedyś program L0phtCrack. Kolejnym krokiem administratora może być przeanalizowanie wartości MRU (Most Recently Used) w poszukiwaniu plików z rozszerzeniem *.lc*. Nawet jeśli użytkownik przezornie usunął plik *sam_pdc.lc* z systemu plików, odwołania do tego pliku nadal mogą się znajdować w Rejestrze!

# Zdobywanie zaszyfrowanych haseł systemu Windows

Po pobieżnym przejrzeniu poprzedniego podrozdziału poświęconego aplikacji L0phtCrack, być może przypuszczasz, że zaszyfrowane kody haseł w systemach Windows mogą być równie łatwo przeglądanego przez administratora, jak plik tekstowy */etc/shadow* w systemach Unix. Z drugiej strony, wykorzystywany w systemach Unix plik tekstowy */etc/shadow* może być przeglądany w najprostszych edytorach tekstu lub po prostu skierowany na standardowe wyjście (ekran). Utrzymywana w systemie Windows baza danych SAM ma nieczytelny format binarny i dodatkowo nie zezwala na tak łatwe kopiowanie czy przeglądanie. Dlatego właśnie potrzebujemy takich narzędzi jak pwdump lub lsadump, które generują na podstawie bazy danych SAM jej wersję tekstową.

## pwdump

Napisany przez Todda Sabina program pwdump2 (patrz strona *http://razor.bindview.com/tools/desc/pwdump2_readme.html*) może być wykorzystywany do wydobywania zaszyfrowanych haseł z systemu Windows. To obsługiwane w wierszu poleceń narzędzie musi być uruchamiane lokalnie w badanym systemie operacyjnym; w dalszej części tego podrozdziału omówimy krótko program pwdump3, który może działać zdalnie.

### Implementacja

Program musi być lokalnie uruchamiany w badanym systemie. Skupimy się teraz na wersji 2. narzędzia, które po raz pierwszy zostało opracowane przez Jeremy'ego Allisona z projektu Samba. Inaczej niż pierwsza wersja, program pwdump2 nie jest ograniczany przez stosowane w bazach danych SAM szyfrowanie SysKey. Metodę szyfrowania SysKey wprowadzono po raz pierwszy w systemie Windows NT — miało to być dodatkowe zabezpieczenie bazy danych SAM, jednak skuteczność tego mechanizmu pozostawia wiele do życzenia, czego dowodem jest choćby sprawne działanie narzędzia pwdump2. Poniżej przedstawiamy sposób korzystania z programu pwdump2:

```
C:\>pwdump2.exe /?

Pwdump2 - dump the SAM database.
Usage: pwdump2.exe <pid of lsass.exe>
```

Aby program mógł wyświetlać zaszyfrowane kody haseł, musi być uruchamiany przez użytkownika z uprawnieniami administratora:

```
C:\>pwdump2.exe

Administrator:500:f1e5c5efbc8cfb7f18136fb05f77a0bf:55c77b761ffa46...
Orc:501:cbc501a4d2227783cbc501a4d2227783:f523558e22c95c62a6d6d00c...
skycladgirl:1013:aa5536a42ebe131baad3b235b51404ee:db31alee00bfbee...
```

Zazwyczaj nie musimy wpisywać w wierszu poleceń identyfikatora procesu (PID), który odpowiada działającemu programowi *lsass.exe*. Jeśli jednak jest to konieczne, możemy wykorzystać kilka prostych sposobów odkrywania tego identyfikatora za pomocą polecenia `tlist` lub `pulist` oraz polecenia `find` (opcja `/i` powoduje, że program `find` ignoruje wielkość znaków):

```
C:\>tlist | find /i "lsass"
 244 LSASS.EXE

C:\>pulist | find /i "lsass"
LSASS.EXE         244  NT AUTHORITY\SYSTEM

C:\>pwdump2.exe 244
Administrator:500:f1e5c5efbc8cfb7f18136fb05f77a0bf:55c77b761ffa46...
Orc:501:cbc501a4d2227783cbc501a4d2227783:f523558e22c95c62a6d6d00c...
skycladgirl:1013:aa5536a42ebe131baad3b235b51404ee:db31alee00bfbee...
```

Jedyną wadą danych generowanych przez program pwdump2 jest brak możliwości ich odczytania przez narzędzie L0phtCrack. Wynika to wyłącznie z faktu, że pwdump2 zwraca zaszyfrowane kody haseł z małymi literami, a program L0phtCrack oczekuje na

wejściu wielkich liter. Warto więc pamiętać, że narzędzie John the Ripper ignoruje wielkość liter.

Możemy na szczęście wykorzystać program tr (translate-character), który przekształci otrzymane dane do postaci obsługiwanej przez łamacza haseł z graficznym interfejsem użytkownika. Program tr jest dostępny w większości systemów Unix i w środowisku Cygwin, został także przeniesiony do systemów Windows jako element zestawu Resource Kit.

```
[user@hedwig]$ cat pwdump.out | tr a-z A-Z
ADMINISTRATOR:500:F1E5C5EFBC8CFB7F18136FB05F77A0BF:55C77B761FFA46...
ORC:501:CBC501A4D2227783CBC501A4D2227783:F523558E22C95C62A6D6D00C...
SKYCLADGIRL:1013:AA5536A42EBE131BAAD3B235B51404EE:DB31A1EE00BFBEE...
```

### pwdump3

Program pwdump3 (*http://www.ebiz-tech.com/pwdump3/*) autorstwa Phila Staubsa jest kolejnym rozszerzeniem narzędzia pwdump. W tej wersji istnieje możliwość uzyskiwania zdalnego dostępu do komputera ofiary. Wersja pwdump3e umożliwia nawet szyfrowanie zdalnych połączeń, dzięki czemu można się zabezpieczyć przed użytkownikami podsłuchującymi komunikację sieciową celem przechwycenia podatnych na atak haseł. Sposób użycia programu pwdump3e różni się nieco od omawianej wcześniej wersji pwdump2:

```
Usage: PWDUMP3 machineName [outputFile] [userName]
C:\>PwDump3.exe victim pwdump.out root
C:\>type pwdump.out
guest:1001:NO PASSWORD**********************:2DEAC3223C70B24E90F02...
wwwadmin:500:NO PASSWORD**********************:9CBD10B05F8E69B62F2...
IUSR_WWW01:1003:6E72211CDC51C9F8EB9293C3135F3985:0E2A2DCE3B6ABFBA...
```

Aby program pwdump3 mógł działać prawidłowo, musi istnieć możliwość ustanowienia połączenia z zasobem ADMIN$. Program ustanawia takie połączenie automatycznie i w odpowiednim momencie wzywa nas do podania hasła administratora. Możemy także ustawić sesję połączenia z ADMIN$ samodzielnie — jak zwykle używamy do tego polecenia `net`:

```
C:\>net use \\victim\admin$ * /u:Administrator
Type the password for \\localhost\admin$:
The command completed successfully.
```

## lsadump2

Program lsadump2 (*http://razor.bindview.com/tools/desc/lsadump2_readme.html*) znacząco upraszcza proces wydobywania haseł z systemu operacyjnego Windows. Jest to kolejne przydatne narzędzie Todda Sabina — powstało przez udoskonalenie oryginalnej aplikacji autorstwa Paula Ashtona. Różnica pomiędzy programem lsadump2 a serią narzędzi pwdump polega na tym, że lsadump2 wykonuje zrzuty rzeczywistych, tekstowych postaci haseł, zamiast generować na wyjściu tylko ich zaszyfrowane wersje. Takie rozwiązanie jest oczywiście bardzo korzystne, ponieważ eliminuje konieczność uruchamiania dodatkowych narzędzi łamiących hasła. Niestety, program lsadump2 potrafi rozszyfrować hasło tylko w sytuacji, gdy zostało ono umieszczone przez mechanizm Local Security Authority (LSA). Może się tak zdarzyć, gdy aplikacja internetowa łączy się z bazami danych SQL, lub kiedy oprogramowanie tworzące kopię zapasową zdalnie łączy się z systemem w celu archiwizowania przechowywanych w nim plików.

### Implementacja

Uruchomienie programu lsadump2 wymaga dostępu do konta administratora. Sposób użycia tego narzędzia przedstawiamy poniżej:

```
C:\>lsadump2.exe
Lsadump2 - dump an LSA secret.
Usage: lsadump2.exe <pid of lsass.exe> <secret>
```

Musimy określić identyfikator PID procesu lsass (robimy to tak samo, jak w przypadku programu pwdump2):

```
C:\>tlist |  find /i "lsass"
244 LSASS.EXE
```


Identyfikator PID procesu LSA jest także przechowywany w Rejestrze, w kluczu *HKLM\SYSTEM\CurrentControlSet\Control\Lsa\LsaPid*.

Narzędzie generuje na wyjściu (w postaci tekstu) tajne dane znajdujących się aktualnie w pamięci procesów związanych z bezpieczeństwem systemu. Może to być hasło wykorzystywane przez konto usługi, informacje o numerze telefonu usługi RAS lub hasło do zdalnego narzędzia tworzącego kopie zapasowe. Dane wyjściowe są umieszczane w dwóch kolumnach:

```
aspnet_WP_PASSWORD
 61 00 77 00 41 00 39 00 65 00 68 00 68 00 61 00  a.w.A.9.e.h.h.a.
 4B 00 38 00                                      K.8.
```

Lewa kolumna reprezentuje surowe wartości szesnastkowe związane z daną usługą. Prawa kolumna zawiera odpowiadającą tym danym reprezentację ASCII. Jeśli zainstalowałeś ostatnio usługę .NET w swoim systemie Windows 2000, najprawdopodobniej został stworzony w systemie użytkownik ASPNET. Jak widać, program lsadump2 bez trudu odkrył hasło tego użytkownika (oznaczone pogrubioną czcionką). Pamiętaj, że system Windows przechowuje hasła w formacie Unicode, co sprawia, że po każdej literze występuje znak pusty (00). Na szczęście, domyślne ustawienia użytkownika ASPNET nie dają mu uprawnień do zdalnego logowania ani do uruchamiania poleceń.

# Aktywne narzędzia atakujące metodą pełnego przeglądu

Aktywne narzędzia są ostatnią deską ratunku dla hakerów, którzy bezskutecznie próbowali zdobyć zaszyfrowane kody haseł. Programy tego typu powodują sporo zamieszania w sieci komputerowej oraz w atakowanych systemach (chociaż przez długi czas mogą działać niezauważenie). Najtrudniejszym elementem na początku aktywnego ataku jest uzyskanie poprawnej nazwy użytkownika systemu docelowego. Więcej informacji na temat technik zdobywania nazw użytkowników znajdziesz w rozdziale 7.

Kolejnym ważnym krokiem przed rozpoczęciem właściwego ataku jest próba odkrycia progu blokującego. Jeśli okaże się, że okres blokowania konta po wprowadzeniu pięciu niepoprawnych haseł trwa 30 minut, nie będziemy oczywiście tracili 29 minut i 30 sekund na podejmowanie prób, które nie mogą doprowadzić do sukcesu.

## THC-Hydra

Program Hydra zdecydowanie przewyższa większość dostępnych w internecie narzędzi do aktywnego łamania haseł metodą testowania wszystkich możliwości z dwóch powodów: jest szybki i może atakować mechanizmy uwierzytelniania dla bardzo wielu różnych protokołów. Niewątpliwą zaletą programu jest dostępność kodu źródłowego (zgodnie z licencją GPL) oraz fakt, że jest częścią narzędzia Nessus.

### Implementacja

Program Hydra można bez trudu skompilować w systemach BSD i Linux; kompilacja aktualnej wersji narzędzia (2.4) jest, niestety, niemożliwa w środowisku Cygwin i systemie Mac OS X. Jeśli pracujesz w systemie BSD lub Linux, wykonaj typowe polecenia `./configure`, `make` i `make install`. Po zakończonej pomyślnie kompilacji możesz zapoznać się z dostępnymi argumentami wiersza poleceń:

```
Hydra v2.4 (c) 2003 by van Hauser / THC <vh@thc.org> Syntax:
hydra [[[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-o FILE]
  [-t TASKS] [-g TASKS] [-w TIME] [-f] [-e ns] [-s PORT] [-S] server
  service [OPT]

Options:
  -S        connect via SSL
  -s PORT   if the service is on a different default port, define it here
  -l LOGIN  or -L FILE login with LOGIN name, or load several logins from FILE
  -p PASS   or -P FILE try password PASS, or load several passwords from FILE
  -e ns     additional checks, "n" for null password, "s" try login as pass
  -C FILE   colon separated "login:pass" format, instead of -L/-P option
  -o FILE   write found login/password pairs to FILE instead of stdout
  -f        exit after the first found login password pair
  -t TASKS  run TASKS number of connects in parallel (default: 4)
  -g TASKS  start TASKS number per second until -t TASKS are reached
  -w TIME   in seconds, defines the max wait reply time (default: 30)
  server    the target server
  service   the service to crack
  OPT       some service modules need additional input, put it here
```

Cel ataku definiujemy za pomocą argumentów `server` i `service`. Typem usługi może być dowolna z wymienionych poniżej aplikacji. Zauważ, że dla wielu z tych usług zdefiniowano już port dla dostępu SSL. Pierwsza liczba w nawiasie reprezentuje domyślny port usługi; druga liczba jest numerem portu dla komunikacji SSL. Jeśli usługa docelowa nasłuchuje na innym porcie, nie zapomnij użyć w wierszu polecenia opcji `-s`. Oto lista usług aktualnie rozpoznawanych przez program Hydra:

- **telnet (23, 992)** — zdalna powłoka poleceń.
- **ftp (21, 990)** — transfer plików.
- **pop3 (110, 995)** — dostęp do poczty elektronicznej.
- **imap (143, 993)** — dostęp do poczty elektronicznej.
- **smb (139, 139)** — usługi SMB systemu Windows, w tym udostępnianie plików i dostęp do IPC$.

- **http (80, 443)** — usługi WWW.
- **https (brak, 443)** — usługi WWW przez SSL.
- **cisco (23)** — typowe dla urządzeń Cisco wezwanie usługi telnet, w której wymagane jest tylko hasło.
- **cisco-enable (23)** — wejście w tryb *enable* (lub *super-user*) urządzenia Cisco. Musimy znać hasło początkowego logowania i podać je w wierszu polecenia po opcji `-m` i bez opcji `-l` czy `-L` (nie będziemy musieli podawać nazwy użytkownika).

```
hydra -m letmein -P password.lst 10.0.10.254 cisco-enable
```

- **ldap (389, 636)** — protokół LDAP (Lightweight Directory Access Protocol) jest często wykorzystywany do pojedynczych rejestracji w systemie.
- **mysql (3306, 3306)** — baza danych.
- **nntp (119, 563)** — dostęp do grup dyskusyjnych USENET.
- **vnc (5900, 5901)** — zdalne administrowanie środowiskami z graficznym interfejsem użytkownika.
- **rexec (512, 512)** — usługa do zdalnego uruchamiania poleceń w systemach Unix. W niektórych systemach dostęp do tej usługi domyślnie nie jest rejestrowany w dziennikach zdarzeń.
- **socks5 (1080, 1080)** — serwer pośredniczący (proxy).
- **icq (4000, brak)** — komunikator internetowy. ICQ wykorzystuje protokół UDP, co oznacza, że nie jest możliwa komunikacja przez SSL.
- **pcnfs (0, brak)** — protokół wykorzystywany do drukowania plików w sieci. Domyślne numery portów różnią się w zależności od dystrybucji systemu i stosowanych serwerów, co oznacza, że zawsze musimy je ustawiać bezpośrednio w wierszu poleceń (za pomocą opcji `-s`). Także ta usługa wykorzystuje protokół UDP, nie jest więc możliwa komunikacja przez SSL.

Uruchamianie programu Hydra jest proste. Największym problemem, z jakim możesz się zetknąć jest wybór właściwej kombinacji nazwy użytkownika i hasła. Poniżej przedstawiamy przykład ataku na usługę SMB systemu Windows. Jeśli na docelowym serwerze jest otwarty port 139. lub 445. i wystąpi błąd, może to oznaczać, że usługa *Server* atakowanego systemu Windows nie została uruchomiona — w takim przypadku nasz atak nie może się oczywiście zakończyć sukcesem.

```
[mike@corrino]$ hydra -L user.lst -P password.lst 10.0.1.4 smb
Reduced number of tasks to 1 (smb does not like parallel connections)
Hydra v2.4 (c) 2003 by van Hauser / THC - use allowed only for legal purposes.
Hydra is starting!  [parallel tasks: 1, login tries: 42 (l:6/p:7)]
[139][smb] login: administrator   password: changeme%
Hydra finished.
```

Hydra wyświetla zarówno łączną liczbę sprawdzanych kombinacji (zwykle jest to liczba unikatowych nazw użytkownika pomnożona przez liczbę unikatowych haseł), jak i liczbę uruchomionych równoległych zadań.


Nigdy nie będziemy mogli uruchamiać więcej niż jednego równoległego zadania przeciwko usłudze SMB, nawet jeśli użyjemy opcji `-t`, która w założeniu zwiększa liczbę równoległych zadań. Z jakiegoś powodu równoległe procesy logowania do usługi SMB powodują dużą liczbę fałszywych wyników negatywnych. Domyślną wartością dla opcji `-t` jest 4, taka liczba równoległych zadań jest zalecana dla urządzeń Cisco oraz serwerów VNC (Virtual Network Computing). Maksymalna liczba zadań wynosi 255, taka liczba nie zawsze jednak zapewnia optymalny czas pracy i najbardziej wiarygodne wyniki.

Jeśli naprawdę chcesz, by przeprowadzane testy były optymalne (żeby tym samym nie były testami wyczerpującymi), powinieneś rozważyć zastosowanie opcji `-C` zamiast wskazywać plik dla każdej opcji `-L` (dla każdej nazwy użytkownika) i dla każdej opcji `-P` (dla każdego hasła). Po opcji `-C` wskazujemy pojedynczą nazwę pliku jako jej argument. Plik ten zawiera kombinacje nazw użytkowników i haseł oddzielonych dwukropkami. Ta metoda często okazuje się bardziej efektywna niż testowanie kont, ponieważ możemy stworzyć plik ze znanymi kombinacjami nazw użytkowników i haseł, co ogranicza liczbę niepotrzebnych prób dla nieistniejących nazw użytkowników. Takie rozwiązanie jest szczególnie przydatne w sytuacjach, w których chcemy testować występowanie wyłącznie domyślnych i najbardziej popularnych haseł.

Kiedy testujesz usługi w swojej sieci, nie zapomnij o wykorzystaniu opcji `-e`. Opcja ta włącza testowanie specjalnych przypadków, czyli braku hasła (`-e n`) lub hasła identycznego z nazwą użytkownika (`-e s`).

**Studium przypadku: sprawdzanie strategii zarządzania hasłami**

Takie narzędzia jak Hydra są wykorzystywane w dwóch sytuacjach: albo podczas ataku na komputery w sieci, albo podczas kontrolowania zabezpieczeń systemów. Oba procesy przebiegają podobnie, jednak istnieje zasadnicza różnica pomiędzy ich celami. Przyjrzyjmy się pracy administratora systemu w dziale wewnętrznego audytu. Pracownicy tego działu nie administrują systemami, sprawdzają jedynie, czy systemy działają zgodnie z przyjętą w firmie polityką bezpieczeństwa. Innymi słowy, do zadań działu należy testowanie kont sieciowych w poszukiwaniu haseł niezgodnych z ustanowionymi zasadami.

Polityka bezpieczeństwa wymaga stosowania ochrony hasłami wszystkich kont (nie mogą istnieć puste hasła), definiowania niebanalnych haseł (interpretacja tego warunku nie jest jednoznaczna, z pewnością jednak hasła nie mogą być takie same jak nazwy użytkowników), zawierania w hasłach przynajmniej jednej cyfry i jednego znaku interpunkcyjnego (nie są dozwolone hasła składające się z samych liter) i tworzenia haseł co najmniej ośmioznakowych. W niektórych systemach Windows i Unix możemy wymusić stosowanie tych reguł w procesie zmiany haseł przez użytkowników. W innych systemach, np. pracujących w urządzeniach Cisco, nie jest to możliwe.

Jednym z zadań działu kontroli jest znajdowanie słabych haseł w jednym z następujących scenariuszy:

- System nie zawiera metod umożliwiających wymuszanie wprowadzania dobrych haseł. Pracownicy działu kontroli muszą w tym względzie zaufać użytkownikom.
- System zawiera metody umożliwiające wymuszanie wprowadzania dobrych haseł, jednak metody te są błędnie skonfigurowane. Użytkownicy powinni realizować politykę zarządzania hasłami, jednak jej wskazania nie są automatycznie wymuszane.
- System zawiera metody umożliwiające wymuszanie wprowadzania dobrych haseł, jednak użytkownicy mogą łatwo omijać zdefiniowane reguły i wprowadzać banalne hasła (np. password99!, pa$$w0rd lub adm1n1str@t0r).

Pracownicy działu zidentyfikowali już kilka usług sieciowych, które mogą się stać łatwym celem dla hakerów. Nie oznacza to jednak, że najlepszym rozwiązaniem w tym momencie jest stworzenie listy wszystkich użytkowników, pobranie z internetu słownika z dwustoma tysiącami słów i uruchomienie programu Hydra (lub wielu procesów tego narzędzia, z uwagi na ilość pracy do wykonania). Zamiast tego można opracować słownik ze słowami, które nie spełniają zaleceń polityki bezpieczeństwa oraz z wyrazami zgodnymi z tą polityką, ale stworzonymi przez proste zastępowanie samogłosek cyframi lub inne podobne zabiegi. Doskonałą metodę tworzenia list haseł na podstawie reguł dotyczących długości i zawartości oferuje program John the Ripper (wspomniany wcześniej w tym rozdziale). Możemy następnie stworzyć (wyłącznie w celach testowych) plik *oldwrods.txt*, w którym umieścimy używane przez administratorów hasła przed ich ostatnią zmianą. Plik *oldwords.txt* ma składnię w postaci: `nazwa_użytkownika:hasło`. Przykładowo:

```
root:web34addmin!
Administrator:thiS1&thaT1
oracle:2bdb|!2bdb
```

Spróbujmy streścić działania działu kontroli. Jego pracownicy stworzyli trzy pliki (i mogą jeszcze stworzyć czwarty i piąty):

- ***users.txt*** — lista wszystkich (znanych) nazw użytkowników systemów podłączonych do sieci.
- ***passwords.txt*** — lista najczęściej spotykanych kombinacji od jednej do siedmiu liter, plus wybrane ośmioliterowe kombinacje z cyframi zastępującymi samogłoski. Większość zawartych w tym pliku wpisów można otrzymać ze słowników dostępnych w internecie, domyślnego pliku *password.lst* dołączanego do programu John the Ripper lub pliku wygenerowanego za pomocą tego narzędzia. Aby ograniczyć liczbę zakończonych niepowodzeniem operacji logowania, które będą rejestrowane przez serwery, na naszej liście nie umieściliśmy więcej niż tysiąc kombinacji.
- ***oldwords.txt*** — lista kombinacji kont i haseł, które zostały zmienione w ciągu ostatnich dziewięćdziesięciu dni. Oczywiście taki plik powinien być należycie zabezpieczony.
- **Puste hasła** — używamy opcji `-e n` w programie Hydra, aby sprawdzić wszystkie konta w poszukiwaniu pustych haseł.
- **„Te same" hasła** — używamy opcji `-e s` w programie Hydra, aby sprawdzić wszystkie konta w poszukiwaniu haseł identycznych z nazwami użytkowników.

Jak dotąd dział kontroli wykonał całkiem sporo pracy nawet bez sprawdzania, czy uda się skompilować program Hydra. Możemy więc śmiało stwierdzić, że pracownicy tego działu stworzyli metodę testowania realizacji opracowanej przez nich strategii zarządzania hasłami. Na tym etapie można już uruchomić program Hydra dla wyselekcjonowanych usług (oczywiście po uprzednim upewnieniu się, że weryfikowane konta nie będą blokowane po nieudanych próbach logowania). Każdy otrzymany w tej fazie wynik pozytywny powinien zwrócić uwagę administratorów sieci i systemu, ponieważ wskazuje na niezgodność z wymaganiami przyjętej polityki bezpieczeństwa.

Wyobraźmy sobie teraz sytuację, w której pracownicy działu wewnętrznej kontroli przystępują do audytu haseł bez odpowiedniego przygotowania — pobierają z internetu słownik wyrazów z dziesięcioma tysiącami pozycji i uruchamiają program Hydra testujący dwieście kont. Jeśli będą mieli szczęście, być może uda im się zakończyć kontrolę w ciągu weekendu. Jeśli będą mieli naprawdę wielkie szczęście, żaden serwer nie ulegnie awarii spowodowanej całkowitym wykorzystaniem dostępnej przestrzeni dyskowej na pliki dzienników, w których rejestrowane są nieudane próby logowania. Co pracownicy działu powinni zrobić, kiedy stosunkowo dobre hasło, np. *ou@te1tw2 lub -#*crAft0,* zostanie odgadnięte tylko dlatego, że znajduje się w użytym słowniku? W takim przypadku przekonanie użytkownika, że naruszył zasady bezpieczeństwa będzie przecież bardzo trudne, ponieważ stworzone przez niego hasło nie jest banalne.

Z drugiej strony, odrobina szczęścia w połączeniu z rozbudowanym słownikiem może skutkować pomyślnym zakończeniem takiego testu sieci komputerowej. Dochodzimy więc do punktu, w którym w procesie kontroli haseł kończy się rola programu Hydra, a zaczyna się rola narzędzia do testów penetracyjnych.