ŁUKASZ CIESIELSKI

Złam hasło!

Stopień trudności

Do niedawna hasła były wystarczającym zabezpieczeniem ważnych danych przed nieuprawnionym dostępem do nich. Czy jest tak nadal? Z całą pewnością nie. Posiadając odpowiednią wiedzę oraz oprogramowanie, użytkownik jest w stanie bez większych przeszkód uzyskać dostęp do zabezpieczonych plików.

Z ARTYKUŁU DOWIESZ SIĘ

jak zdobyć hasło dowolnego konta w systemie Linux,

jak posługiwać się najpopularniejszym programem służącym do łamania haseł – John the Ripper,

w jaki sposób testować bezpieczeństwo haseł,

co zrobić, aby przeczytać zabezpieczony plik PDF,

jak rozpakować archiwum ZIP bez znajomości hasła dostępu,

w jaki sposób uzyskać dostęp do konta dowolnego użytkownika systemu Windows XP i Vista.

CO POWINIENEŚ WIEDZIEĆ

umieć sprawnie posługiwać się konsolą,

znać metody łamania haseł (Single Crack, słownikowa, brute-force),

znać metody kodowania haseł (DES, RSA, MD4, MD5, Kerberos AFS itp.).

Artykuł wprowadza w arkana świata crackingu i ukazuje sposoby łamania haseł. Już po kilku chwilach znajomość metod postępowania z zabezpieczonymi plikami i hasłami do systemów operacyjnych może okazać się nie tylko bardzo pożyteczną umiejętnością, ale także doskonałą zabawą, a nawet przygodą.

Na skróty

Obiektem naszych zainteresowań są tzw. funkcje skrótu (dokładniej omówione zostały poniżej). Dlaczego właśnie one? Ponieważ dzięki nim możemy dowiedzieć się, jakie hasło zostało przyporządkowane danemu użytkownikowi. Aby jednak zrozumieć istotę sprawy, musimy sprawdzić, czym charakteryzują się najczęściej wykorzystywane algorytmy, takie jak MD5, SHA-1 itd.

Co można powiedzieć o algorytmie MD5 (ang. *Message-Digest algorithm*)? Najważniejszą cechą jest oczywiście długość generowanego skrótu, czyli 128 bitów. Pomysłodawcą jest profesor Ronald Rivest. Został on zmuszony do opracowania kolejnej wersji, po tym jak Hans Dobbertin udowodnił, że poprzedni algorytm, MD4, jest zbyt słaby i nie daje gwarancji bezpieczeństwa. Prototyp algorytmu MD5 powstał w 1991 roku. Pięć lat później po raz kolejny Hans Dobbertin starał się obalić mit o niezawodności MD5. Choć jego analiza nie dostarczyła dostatecznie przekonujących dowodów na temat słabości algorytmu Ronalda Rivesta, wystarczyła jednak, aby eksperci z zakresu kryptografii i bezpieczeństwa częściowo zrezygnowali z MD5 na rzecz SHA-1 oraz RIPEMD-160. 17 sierpnia 2004 grupa chińskich naukowców: Xiaoyun Wang, Dengguo Fen, Xuejia Lai i Hongbo Yu opublikowała analityczny algorytm ataku, niepodważalnie wykazując słabość MD5.

Szyfrowanie algorytmem MD5 przebiega w sześciu fazach. Na początku do wartości wejściowej zostaje doklejony bit o wartości 1. Następnie wynik jest uzupełniany zerami do momentu, aż ciąg będzie składał się z 512-bitowych bloków, a ostatni z nich osiągnie długość 448 bitów. Ostatni blok musi być niepełny, ponieważ to właśnie do niego zostaje doklejony w kolejnej fazie 64-bitowy licznik, określający rozmiar wiadomości. Teraz już wiadomość składa się w całości z 512-bitowych fragmentów. Kolejnym krokiem jest ustawienie wartości początkowej na *0123456789abcdeffedcba9876543210*. Po tych czynnościach na każdym bloku zostaje uruchomiona funkcja zmieniająca stan. Po zmianie stanu ostatniego bloku zostaje zwrócony gotowy skrót wiadomości.

Drugim równie często stosowanym algorytmem jest SHA-1 (ang. *Secure Hash Algorithm*). Został on zaprojektowany przez NSA (ang. *National Security Agency*) po ujawnieniu słabych punktów MD5 i SHA-0. SHA-1 został opublikowany w 1995 roku przez National Institute of Standards and Technology. Niestety, okazało się, że i ten algorytm nie zdał

egzaminu. Od 2002 roku powstały cztery warianty SHA-2, czyli SHA-224, SHA-256, SHA-384 i SHA-512. Choć w 2004 roku oficjalnie ogłoszono udane ataki na strukturę podobną do SHA-1, w dalszym ciągu jest to algorytm powszechnie stosowany. National Institute of Standards and Technology dopiero w 2010 roku planuje całkowitą rezygnacje z SHA-1 na rzecz nowszego SHA-2. Jeżeli chodzi o stronę techniczną SHA-1, to jest on bardzo podobny do MD5, z tą różnicą, że tworzy 160-bitowy skrót wiadomości. Najpopularniejszym zastosowaniem dla SHA jest zabezpieczanie podpisów cyfrowych, baz danych MySQL, dokumentów Worda i Excela. Często wykorzystuje się ten algorytm również w kryptografii, a nawet przemyśle i organizacjach wojskowych.

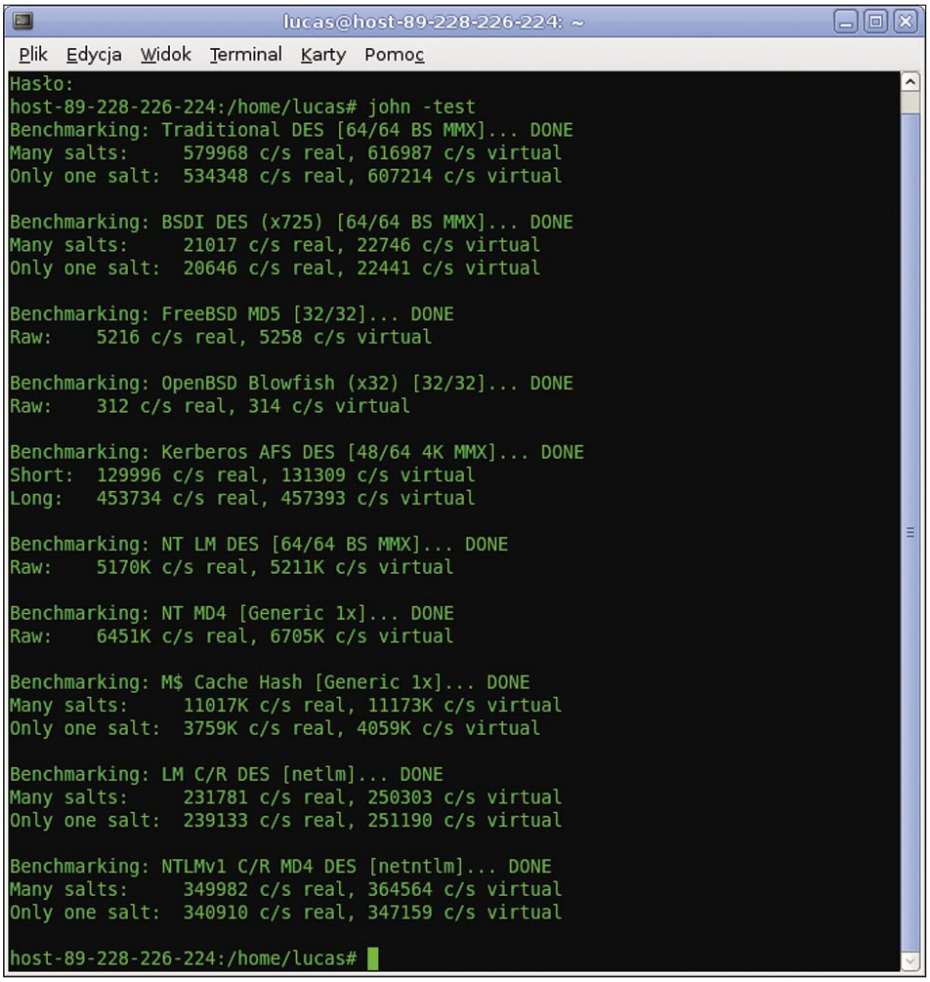
Najlepszy z najlepszych

Łamanie haseł jest czynnością wymagającą oprogramowania najlepszego z najlepszych. Właśnie ta myśl przyświecała twórcy programu John the Ripper. Jego autorem jest Alexander Peslyak, znany pod pseudonimem Solar Designer. Jako nie tylko programista, lecz również doskonały haker, wiedział, na czym polega deszyfrowanie zakodowanych plików i ciągów znakowych. Warto wspomnieć, że poza programem służącym do łamania haseł, Alexander Peslyak opracował dystrybucję łat na jądro Linuksa – Openwall, w której maksymalny nacisk został położony na bezpieczeństwo w każdej postaci. Wróćmy jednak do programu John the Ripper, aby rozpocząć przygodę z deszyfrowaniem i hasłami w najrozmaitszej postaci.

Instalacja nie powinna sprawić kłopotu, ponieważ John the Ripper znajduje się w repozytorium wielu dystrybucji systemów operacyjnych. Aktualna wersja – 1.7.3.1 – została wydana 18 lipca 2008 roku. Ogromną zaletą (poza funkcjonalnością) jest wieloplatformowość. John the Ripper został udostępniony na takie systemy jak: UNIX, Linux, Windows, BeOS, OpenBSD, FreeBSD, IRIX, AIX, a nawet HP-UX. Biorąc pod uwagę fakt, że nie jest to program napisany w środowisku Java, mnogość dostępnych platform budzi pewien podziw. Po instalacji warto sprawdzić, czy wszystko zostało poprawnie skonfigurowane. Najlepszym sposobem jest uruchomienie programu z parametrem *-test*.

Dzięki temu program obliczy wartości *c/s* (ang. *cracs per second*), czyli ilość prób rozszyfrowania hasła na sekundę dla poszczególnych systemów operacyjnych. Jeżeli John the Ripper przeprowadził test bez problemów, oznacza to, że jest gotowy do pracy.

Zanim przystąpimy do łamania plików z hasłami, należy przypomnieć, że program ten potrafi pracować z określonymi algorytmami. Najczęściej są one związane z plikami przechowującymi hasła dostępu do systemów operacyjnych, np. */etc/passwd, /etc/shadow* (Linux) lub *\WINNT\repair\SAM*. Aby zdobyć hasło (zakodowane) w systemie Windows, należy skorzystać z programu *pwdump*. John the Ripper bez problemów odczytuje niezbędne informacje z wymienionych powyżej plików oraz wyniki działania programu *pwdump*. Nie oznacza to jednak, że za pomocą tego oprogramowania nie można złamać zabezpieczenia np. aplikacji czy serwera. Wręcz przeciwnie, jedyną przeszkodą może być brak wiedzy na ten temat. Niezwykle istotną kwestią jest format ciągu, który chcielibyśmy pozyskać w postaci rozkodowanej. Aby przyjrzeć się składni tzw. hasha, najlepiej edytować plik */etc/shadow* lub */etc/passwd*. W tym momencie powinniśmy ujrzeć charakterystyczny ciąg, np. `lucas:$1$CDjnm/BE$OaPrZUSMDtbUa3mlY9fqK0:14110:0:99999:7::`. Dla osoby, która ma zamiar pozyskać hasło dostępu danego użytkownika, najistotniejsze są dwa pierwsze pola, czyli nazwa użytkownika oraz hasło. Z podanego przykładu wynika, że właścicielem konta jest osoba o loginie *lucas* i haśle... No właśnie – i tu wkracza John the Ripper. Jak łatwo się domyślić, dla uzyskania dostępu do konta administratora, należy szukać linijki zawierającej login *root*. Podobnie będzie wyglądać sytuacja, w której obiektem naszych zainteresowań jest konkretna



Rysunek 1. *Program John the Ripper uruchomiony z parametrem -test*

aplikacja. W takim wypadku wystarczy w miejsce konta użytkownika odnaleźć nazwę konta danej aplikacji i złamać skojarzony z nią skrót.

Jeśli chcemy rozpocząć pracę, korzystając ze standardowych ustawień programu, wystarczy wywołać go w linii poleceń, jako argument podając jedynie ścieżkę do pliku zawierającego wartości mające zostać rozszyfrowane. John the Ripper rozpocznie pracę, co zauważymy natychmiast, ponieważ niemalże od razu obciążenie procesora będzie wynosić sto procent. Algorytm wykorzystany do kodowania hasła zostanie wykryty przez program automatycznie. Niewielkim utrudnieniem jest sposób wyświetlania komunikatów o postępie pracy programu. Najprawdopodobniej ma to związek z dużym obciążeniem procesora. Otóż aby dowiedzieć się, jak postępują prace, należy wcisnąć dowolny przycisk na klawiaturze.

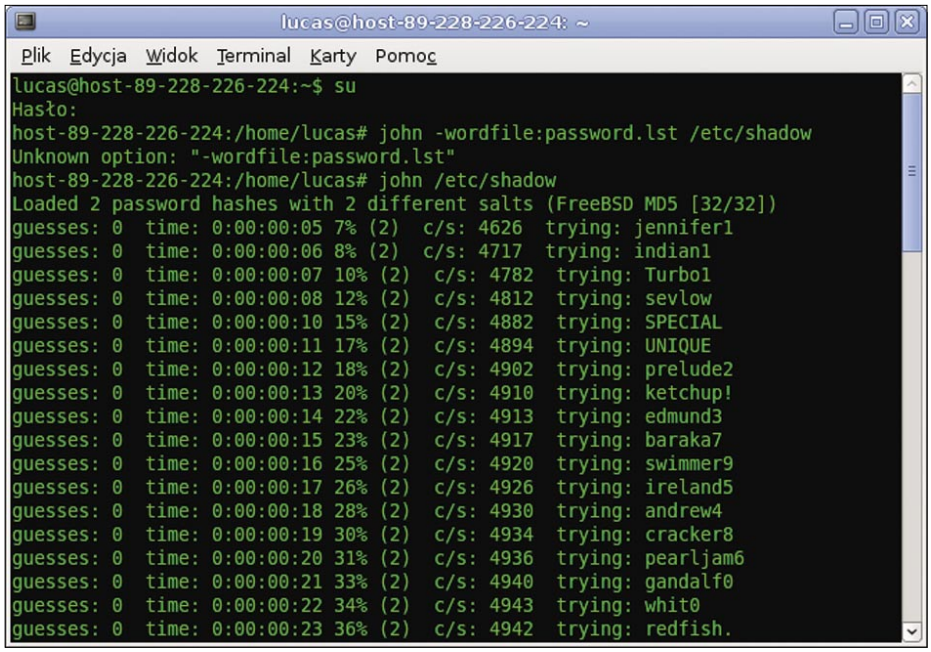
Proces analizy może trwać od kilku minut do kilku dni, a nawet tygodni (a dla naprawdę silnych haseł jeszcze o wiele dłużej!). Jeżeli uznamy, że sprzęt, na którym poddaliśmy analizie dany ciąg znaków, jest zbyt słaby, możemy przerwać działanie programu, korzystając ze skrótu [*Ctrl+c*]. Wysokie użycie mocy obliczeniowej procesora wynika ze specyfiki działania narzędzia.

W rzeczywistości oprogramowanie tego typu składa się z dwóch podstawowych elementów: klienta oraz słownika. Deszyfracja hasła jest w rzeczywistości – przy zastosowaniu metody słownikowej – porównywaniem poszczególnych elementów słownika (są to nie tylko wyrazy, lecz także najczęściej stosowane zestawienia literowe, używane jako hasła) z interesującym nas ciągiem. Jeżeli operacja zakończy się sukcesem, efekt działania programu – czyli wyjściowa postać hasła – zostanie zapisany w pliku *john.pot*. Załóżmy, że chcemy dowiedzieć się, jakie hasła mają użytkownicy danego systemu operacyjnego, np. Debiana. Wiemy już, że aby rozpocząć pracę, wystarczyłoby polecenie `john /etc/shadow`. Metoda ta posiada jednak poważną wadę – jest powolna. Można przyspieszyć działanie programu poprzez dobór konkretnego słownika. Służy do tego opcja `-wordfile:` . Przykładowe polecenie, wskazujące jednocześnie konkretny słownik, który ma zostać użyty przez program, może wyglądać tak:

```
john -wordfile:password.lst /etc/shadow
```

Na początku John the Ripper próbuje złamać hasła, uwzględniając algorytm odpowiadający temu, którym zostało zaszyfrowane hasło. W tym przypadku będzie to *MD5*. Po zakończeniu procesu szukającego hasła, wyniki działania programu zostaną zapisane we wspomnianym wcześniej pliku *john.pot*. Pamiętajmy jednak, że parametry programu przedstawione w powyższych przykładach to jedynie podstawowe opcje. Praktyka i rzeczywistość pokażą, że dla złamania haseł najczęściej będziemy zmuszeni korzystać z opcji zaawansowanych, takich jak np. reguły. To właśnie dzięki regułom program może dokonać permutacji zawartości słownika wykorzystywanego aktualnie przez program. Jeżeli chciałbyś korzystać z innego słownika niż te, które standardowo oferuje John the Ripper, warto odwiedzić strony *www.openwall.com* lub *coast.cs.purdue.edu*. Znajdziemy tam nie tylko niewielkie słowniki, zawierające jedynie hasła skojarzone z określonymi dziedzinami, lecz także duży zbiór (około 40 MB po dekompresji), będący kompilacją wszystkich słowników.

Definicje reguł znajdują się w pliku *john.conf*. Aby określić zasady, na podstawie których John the Ripper będzie wyszukiwał odpowiedników w słowniku, należy posłużyć się symbolami i formatem zaczerpniętymi z programu crack Aleca Muffeta. Pewnie zastanawiasz się, do czego właściwie może przydać się definiowanie reguł, skoro korzystamy ze słownika. Otóż problem polega na tym, że nawet najbardziej rozbudowany słownik nie zawiera wszystkich (także tych najdziwniejszych) możliwych odmian danego wyrazu. Wystarczy, że użytkownik dołączy do danego słowa jakąś cyfrę i hasło nie zostanie odkryte, ponieważ podany wzorzec okaże się nieprawidłowy. I tu do akcji wkraczają wspomniane reguły. Pozwalają one na bardzo szczegółowe definiowanie ostatecznej postaci wyszukiwanych wyrazów. Np. zwykła metoda słownikowa nie odnajdzie słowa *1!pass!1*. Jeżeli jednak określimy pewne zasady wyszukiwania, to bardzo możliwe, że hasło podobnej postaci zostanie odnalezione. Załóżmy, że musimy złamać hasło (mamy po temu ważne powody) i wiemy dwie rzeczy. Po pierwsze, hasło rozpoczyna cyfra z zakresu od 0 do 3, a po drugie – na końcu znajduje się znak procenta. Oczywiście i w tym przypadku możliwości jest bardzo wiele, jednak jeżeli ściśle zdefiniujemy znaki, które mogą znajdować się na początku i na końcu naszego hasła, zbiór możliwych



Rysunek 2. *Przykład łamania hasła z wykorzystaniem metody słownikowej*

rozwiązań zostanie znacznie zawężony. Dodając do tego kolejne informacje, naprowadzamy program na właściwy trop.

Przejdźmy jednak do praktyki. Aby określić, że początkiem szukanego hasła są cyfry od 0 do 3, należy w pliku *john.conf* wpisać regułę `^[0123]`. Znak ^ (*daszek*) oznacza, że następująca po nim definicja wzorca odnosi się do początku poszukiwanego ciągu znaków. W nawiasie kwadratowym wskazujemy zakres znaków, które mają być brane pod uwagę w danym wyszukiwaniu. Oczywiście nie muszą to być jedynie cyfry. Równie dobrze mogliśmy zastosować litery, np. `^[abc]`. Jeżeli za bazę, którą sprawdza John the Ripper, przyjmiemy słowo world, to dla powyższej definicji program utworzy takie hasła jak: *0world*, *1world*, *2world* oraz *3world*. Jak łatwo się domyślić, przy użyciu drugiej reguły wynikiem będą słowa: *aworld*, *bworld* i *cworld*. W identyczny sposób możemy określić takie znaki jak wykrzyknik czy pytajnik.

No dobrze, a co się stanie, jeżeli dodatkowe znaki chcemy przypisać na końcu wzorca? Należy wtedy skorzystać z symbolu `$` (dolar), odnoszącego się do końca badanego ciągu znaków. Dla znaków niebędących ani literami, ani cyframi definicja może wyglądać np. w taki sposób: `$[!@#$%^&*()]`. Dla wzorca *world* John będzie tworzył takie hasła jak *world!*, *world#* czy *world&*. Jest to bardzo przydatne w momencie, kiedy mamy zamiar łamać hasła wygenerowane przez specjalne, stworzone w celu budowania silnych haseł, programy. To jednak nie wszystko. Szybko można dojść do wniosku, że dobre hasło otrzymamy, modyfikując nie tylko początek i koniec danego wzorca, ale także jego wnętrze. Dla wspomnianego już słowa *world*, może to być np. *w0r1d*. Litery o i l zostały zamienione na cyfry 0 i 1. Standardowe wyszukiwanie, polegające na *ślepym* porównywaniu określonego wyrazu z zawartością słownika na nic się nie zda. Jednak John the Ripper oferuje parametr, dzięki któremu wskażemy programowi, że na drugim miejscu w słowie występuje właśnie cyfra 0, natomiast kolejną definicją określimy cyfrę 1 stojącą na czwartym miejscu. Do pliku zestawiającego nasze reguły dopisujemy kolejno `i[2][0]` oraz `i[4][1]`. Jeżeli nie jesteśmy pewni, jaki znak znajduje się na np. 3 miejscu, ale wiemy, że jest to a, b lub c, to bez kłopotu możemy to zdefiniować za pomocą tego samego parametru `i`. Gotowa reguła będzie wyglądała tak: `i[3][abc]`. Dla programu polecenie takie będzie równoznaczne ze sprawdzeniem danego wzorca z podstawieniem na trzeciej pozycji kolejno znaków a, b i c.

Również w pliku *John.conf* użytkownik może określić dodatkowe reguły. Aby jednak którakolwiek z nich została zinterpretowana przez program, należy rozpocząć definiowanie znakiem dwukropka. Następnie trzeba podać pożądane opcje. Kiedy wyszukujemy słów pisanych jednocześnie małymi i dużymi literami, dobrze jest ustawić parametr `-c`. Podane po nim polecenia nie zostaną wykonane, kiedy wyraz będzie pisany jednolicie – małymi lub wielkimi literami, natomiast będą one aktywowane, jeśli w przetwarzanym wzorcu występują oba rodzaje liter. Bardzo często wiemy mniej więcej, jakiej długości słowa poszukujemy. Nie ma sensu, żeby John szukał w słowniku ciągów składających się z dziesięciu znaków, skoro wiemy, że hasło nie zawiera ich więcej niż np. 4.

Przy budowie reguły korzystamy wtedy ze znaku większości, po którym określamy maksymalną długość ciągu (›4). Wpisując natomiast literę c, nakażemy, aby program zamienił wyraz na pisany wielkimi literami (przydatne dla haseł pisanych z włączonym *Caps Lockiem*). Czasami prościej jest zdefiniować ciągi, które mają zostać pominięte podczas porównywania ze słownikiem. Oczywiście jest to możliwe. Chęć zdefiniowania reguły zawierającej zestaw znaków, które nie mają być uwzględniane, wyrażamy poprzez wpisanie `!?` i nazwy klasy, którą wyłączamy.

Metoda słownikowa nie jest jednak domyślnym ustawieniem. Jeżeli nie podamy nazwy słownika, z którego chcemy korzystać, John będzie próbował złamać hasło tzw. metodą *brute-force*. Polega ona na próbowaniu wszystkich możliwych ciągów znaków, tak długo, aż aktualny ciąg okaże się zgodny z rozszyfrowywanym. Niestety, metoda ta jest bardzo czasochłonna. W porównaniu jednak do pracy ze słownikiem okaże się, że to właśnie metoda brute-force jest skuteczniejsza. Dlaczego? Ponieważ każdy, nawet najbardziej rozbudowany słownik jest ograniczony pewnymi ramami. Stosując

Listing 1. *Program haslo.c, sprawdzający stopień skomplikowania hasła użytkownika na podstawie porównań ze słownikiem*

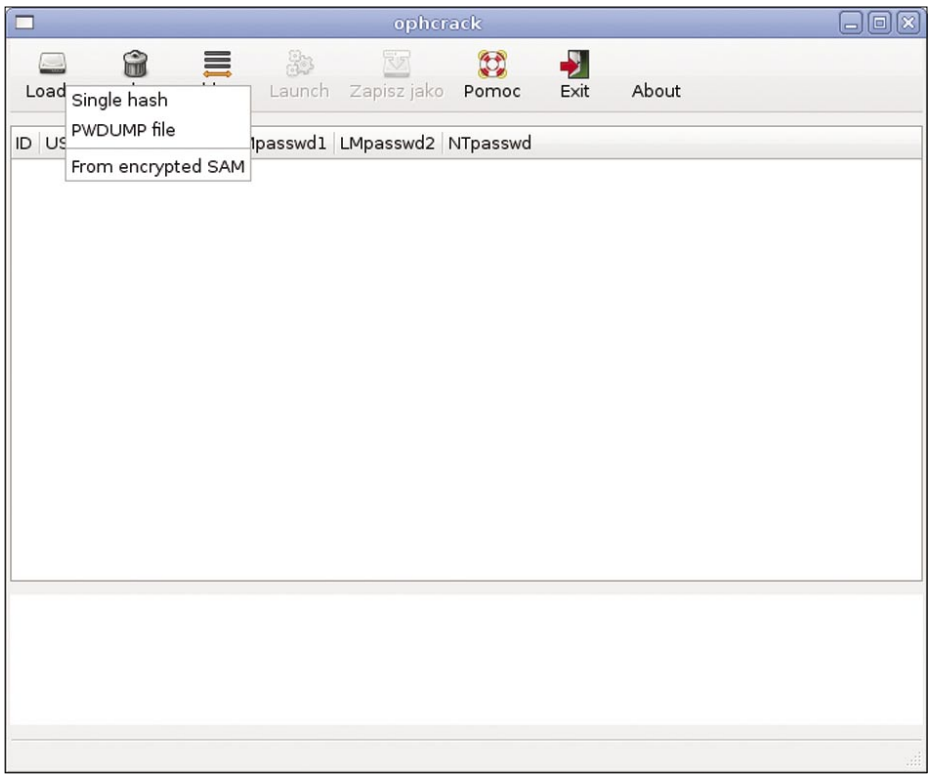
```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <crack.h>
#define slownik "/sciezka/do/katalogu/slownika"
int main(int argc, char*argv[]) {
    char *haslo;
    char *ostrzezenie;
    int status = 0;
    while ((haslo = getpass("\nHasło: ")) != NULL && *haslo ) {
    if ((ostrzezenie = FascistCheck (haslo, slownik)) != NULL) {
        printf("UWAGA!: %s.\n", ostrzezenie);
        status = 1;
    } else {
       printf("System bezpieczny! Hasło jest odporne na złamanie!\n");
    }
}
exit(status);
}
```

Listing 2. *Słowo hakin9 zakodowane algorytmami: MD2, MD4, MD5, SHA-1 oraz RIPEMD*

```
MD2: 6003a4c61683cfc5bf77b6172b0d1289
MD4: 5ffd4ebd37ea2f6c7dc81ea4de964169
MD5: 4156d9df0f672ddf7721796d3e3cd058
SHA-1: 767bbb5e6633d4c48f5af5a3cadd83b781e24d4e
RIPEMD-160: 0e97cf97739ee564826306b239854983248c502a
```

technikę *brute-force*, program pomija ten problem, ponieważ buduje dowolne możliwe ciągi, nie sugerując się gotowym zestawieniem słów.

Jak już wspomnieliśmy, metoda *brute-force* ustawiona jest w programie John the Ripper jako opcja domyślna. Nie oznacza to jednak, że każdy parametr narzędzia został skonfigurowany dla osiągnięcia maksymalnej wydajności. Najczęściej jest właśnie tak, że to użytkownik musi zadbać o ostateczną konfigurację. Aby określić szczegółowe zasady postępowania podczas łamania hasła, niezbędne będzie odpowiednie ustawienie parametru `-incremetal`, np. `john -incremetal:All shadow`. Parametr `-incremetal` może występować w czterech formach. Pierwsza postać to dookreślenie `All`, powodujące, że program weźmie pod uwagę zarówno duże, jak i małe litery oraz wszelkie znaki interpunkcyjne. Jeżeli jesteśmy pewni, że hasło składa się jedynie z małych liter, wystarczy opcja `Alpha`. Podobnie jest w przypadku cyfr (`Digits`). Może się jednak zdarzyć, że będziemy chcieli wyeliminować wszystkie duże litery. Korzystamy wtedy z dodatkowego parametru `LanMan`, dodatkowo odpowiednio go konfigurując. Wbrew pozorom, jest to niezwykle istotna kwestia. Biorąc pod uwagę fakt, że polecenie `-incremetal:All` może zaowocować nawet trylionami rozwiązań, należy w miarę możliwości uściślić nasze oczekiwania względem programu. Dla każdej z wymienionych wcześniej opcji (`All`, `Alpha`, `Digits` oraz `LanMan`) należy zdefiniować w pliku *john.conf* pięć pól, które określają zakres i sposób działania programu. Na początek określamy, która opcja jest niżej definiowana, np. `[incremental:LANMan]`. Kolejnym istotnym krokiem jest wskazanie pliku zawierającego listę dostępnych znaków (`File = ./LANMan.chr`). Teraz możemy już wskazać minimalną i maksymalną długość generowanych ciągów. Dokonujemy tego za pomocą parametrów `MinLen` i `MaxLen`, np. `MinLen=0`. Na koniec nie pozostało nic innego, jak zdefiniowanie liczby znaków dostępnych na liście (`CharCount=100`). Dlaczego parametry te są takie ważne? Ponieważ w dużej mierze to właśnie od nich zależy czas i skuteczność działania programu. Dzięki ogranicznikom, takim jak `MinLen` i `MaxLen`, można skrócić czas łamania hasła (jednocześnie zwiększając prawdopodobieństwo, że... nie uda się go znaleźć). Być może komuś te wszystkie zabiegi wydają się bezcelowe, jednak to twierdzenie może zostać obalone jeszcze przed pierwszą próbą deszyfracji hasła. Wystarczy w prosty sposób obliczyć ilość możliwych kombinacji dla wybranych ustawień. Załóżmy, że nasza maksymalna długość ciągu ma wynosić 5, natomiast na liście będzie 90 znaków. Aby dowiedzieć się, ile różnych ciągów utworzy John podczas sprawdzania, należy parametr `CharCount` podnieść do potęgi `MaxLen` (905). Wynik, mimo niewielkiej długości hasła, będzie olbrzymi. Odpowiednio manipulując tymi ustawieniami, możemy go zmniejszyć nawet o połowę. Nie bez znaczenia okaże się nasza intuicja i umiejętność przewidzenia poczynań właściciela hasła.



Rysunek 3. *Program ophcrack z charakterystycznym i przyjaznym GUI*

Rainbow tables

Chociaż współcześnie większość dystrybucji systemów typu Linux czy BSD wykorzystuje do ochrony haseł użytkowników tzw. sól (ang. *salt*), warto wziąć pod uwagę także bazę skrótów, nazywaną często tęczowymi tablicami (ang. *rainbow tables*). Rozpocznijmy jednak od wyjaśnienia pojęcia wspomnianej soli. Uogólniając, sól to wartość generowana losowo, a następnie dodawana do funkcji skrótu w charakterze jej argumentu i zapisywana obok wartości skrótu. Dzięki takiemu rozwiązaniu wynik funkcji skrótu jest inny dla każdej wartości soli. W praktyce sprawia to, że osoba łamiąca hasło musiałaby posiadać tęczową tablicę dla każdej soli. Oczywiście w takim wypadku korzystanie z tęczowych tablic jest nieuzasadnione i niepraktyczne. Zastanawiasz się pewnie, czym jest tajemnicza funkcja skrótu i tablice o nazwie rodem z baśni? Jak pewnie zauważył każdy, kto choć raz wypróbował program John the Ripper, jego zdecydowanie największą wadą (i zaletą jednocześnie) jest całkowite wykorzystanie mocy obliczeniowej procesora. Efektem jest brak możliwości wygodnego korzystania z innych aplikacji. Zakładając, że przez kilka dni nie mamy zamiaru korzystać z komputera, nie będzie to dla nas problemem. W

przeciwnym wypadku będzie zależało nam na jakiejś innej metodzie, która maksymalnie przyspieszy proces łamania hasła. Tu właśnie wkraczają tęczowe tablice i wspomniana wcześniej funkcja skrótu. Jak wiadomo i łatwo się domyślić, przechowywanie haseł w postaci niezaszyfrowanego tekstu nie miałoby sensu, a przejęcie kontroli nad systemem zajęłoby wprawnemu użytkownikowi kilka sekund. To właśnie z tego powodu powstały funkcje skrótu, takie jak chociażby MD2, MD4, MD5, SHA-1, NTLM, RIPEMD czy Cisco Pix. Niedługo po tym, jak wdrożono nowy pomysł w życie, zaczęto się zastanawiać, w jaki sposób można ominąć takie zabezpieczenie. Dla jednych rozwiązanie tego problemu miało pomóc w dalszych badaniach nad doskonaleniem i zwiększaniem bezpieczeństwa. Znalazła się jednak również grupa osób, którzy chcieli rozwiązać zagadkę, aby móc omijać zabezpieczenia. W ten oto sposób narodził się pomysł na stworzenie swoistej bazy skrótów, którą będzie można wykorzystywać do łamania haseł kodowanych właśnie jednokierunkową funkcją skrótu. Pionierem w tej dziedzinie był Philippe Oechslin. Jak widać na przykładzie słowa *hakin9* (Listing 2), wynik funkcji hashujących jest całkowicie niezrozumiały dla człowieka. Jeżeli jednak wygenerujemy każdą możliwą kombinację postaci zahashowanej, będziemy w stanie porównać tę wartość z kluczem. To właśnie zbiór wszystkich możliwości tworzy tęczowe tablice.
O ile metoda słownikowa daje około 40% szansy na złamanie hasła, to wykorzystanie tęczowych tablic zwiększa szansę do nawet 95%! Dodatkowo czas potrzebny na złamanie hasła tą metodą jest o około jedną czwartą krótszy niż w przypadku pozostałych sposobów. Należy jednak mieć na uwadze fakt, że *rainbow tables* to metoda dająca nie tylko korzyści, ale także będąca w znacznym stopniu uciążliwą – a to ze względu na ich objętość. Rozmiar odpowiedniej tablicy może wahać się od 160 MB do nawet 60 GB! Jeżeli ktoś naprawdę chce korzystać z tęczowych tablic, ma dwie możliwości. Pierwsza to wygenerowanie własnych tablic dla konkretnych wartości. Natomiast druga to pobranie z Internetu już przygotowanych baz zawierających gotowe kombinacje.

Biblioteka crackera

Podobnym działaniem cechuje się program *crack*, bazujący na doskonałej bibliotece *cracklib*. Właściwym zadaniem tego programu i biblioteki (ponieważ to właśnie ona jest filarem całości) nie jest łamanie haseł – tak jak ma to miejsce w przypadku *Johna the Rippera*. Celem *crackliba* jest sprawdzanie wytrzymałości i stopnia skomplikowania danego łańcucha znaków (czyli hasła) za pomocą najróżniejszych mechanizmów. Pewnie zastanawiacie się, dlaczego poruszamy temat związany z tworzeniem haseł. Otóż aby dobrze zrozumieć sposoby omijania zabezpieczeń, należy mieć świadomość, jakie elementy wpływają na stopień bezpieczeństwa oferowanego przez konkretny środek ochrony. Podobnie jest w naszym przypadku. Nie wystarczy umieć skonfigurować oprogramowanie, które wykona całą pracę za nas. Trzeba wiedzieć, który z określonych ciągów jest bezpieczniejszy i jakich komplikacji należy się spodziewać. Wiedza taka pozwoli nie tylko na odpowiedni dobór metody deszyfracji dla programu, ale także da możliwość przybliżonego oszacowania czasu potrzebnego na odnalezienie właściwego kodu dostępu.

Autorem biblioteki jest Alec Muffett, jednak od wersji 2.8 za rozwój odpowiada Nathan Neulinger. Powodem stworzenia *crackliba* stało się ogromne zapotrzebowanie na bibliotekę, pozwalającą zbadać odporność haseł na ataki. Nie bez znaczenia jest również fakt, że za korzystanie z *crackliba* użytkownik nie zostaje obciążony żadnymi kosztami. Funkcjonalnością i zakresem działań biblioteka do złudzenia przypomina program *John the Ripper*. Różnica polega na tym, że na bazie *crackliba* jesteśmy w stanie napisać własny program, który będzie potrafił przeprowadzać testy na szyfrowanych ciągach znaków. Aktualnie biblioteka ta wraz z programem *crack* zostały umieszczone w repozytoriach dużej części dystrybucji Linuksa. Z całą pewnością są dostępne dla Debiana (i wszystkich pochodnych) oraz Slackware. Po instalacji należy zadbać o umieszczenie w odpowiednim katalogu bogatych słowników. Po tych czynnościach program będzie gotowy do pracy. Procesem instalacji nie będziemy się szerzej zajmować, ponieważ nie jest on skomplikowany i nie miałoby to większego sensu. No dobrze, ale w jaki sposób zmusić bibliotekę do pracy? W tym celu musimy napisać własny program. Podany przykład został stworzony w języku *C*, jednak *crackliba* z powodzeniem można wykorzystać w takich językach, jak *Perl* czy *Python*. Najważniejszą funkcją całej biblioteki jest *FascistCheck*. To właśnie dzięki niej program będzie mógł w pewnym sensie zweryfikować jakość tworzonego hasła. Działanie to opiera się na porównaniu podanego przez użytkownika ciągu znaków z wyrażeniami zgromadzonymi w słowniku. Jeżeli ciąg taki zostanie odnaleziony, program wskaże ostrzeżenie, że hasło jest proste do złamania, ponieważ można tego dokonać metodą słownikową (która, jak już wspomnieliśmy, jest z założenia mniej skuteczna od *brute-force*). Użytkownicy powinni się wystrzegać tak słabych zabezpieczeń, natomiast to właśnie na nie czyhają osoby chcące dostać się do cudzych systemów operacyjnych. Przykładowe wykorzystanie biblioteki *cracklib* przedstawia Listing 1. Aby skompilować zamieszczony tam kod, należy użyć polecenia `gcc haslo.c -lcrack -o haslo`. Dzięki temu użytkownik jeszcze przed ustawieniem określonego hasła ma możliwość sprawdzenia go i ewentualnego skorygowania w celu zwiększenia jego odporności na ataki. Osoba zajmująca się łamaniem haseł może natomiast wykorzystać ten niewielki program, aby sprawdzić, jakie hasła mogą stanowić problem i jakie zbitki znaków są dla oprogramowania deszyfrującego hasła najtrudniejsze do rozgryzienia. Wiedza taka może się niejednokrotnie przydać. Należy mieć jednocześnie świadomość, że nawet najbardziej skomplikowane hasło można (przy założeniu dysponowania odpowiednią ilością czasu) złamać i nie daje ono stuprocentowej pewności.

Zagrożenie dla ZIP

W dzisiejszych czasach archiwa takie, jak ZIP czy RAR nie są bezpieczne – nawet jeżeli zostaną zabezpieczone

hasłem. Dystrybucje Linuksa oferują niewielki program napisany przez Marca Lehmanna – *fcrackzip*. Umożliwia on złamanie w prosty sposób zabezpieczeń wskazanego archiwum lub grupy archiwów. Jego działanie jest niemal identyczne jak w przypadku omówionych wcześniej narzędzi. Na podstawie analizy słownika lub przy pomocy metody *brute-force*, *fcrackzip* poszukuje odpowiedniego hasła. Dedykowane pakiety znajdują się w repozytoriach poszczególnych dystrybucji. Jeżeli z jakichś względów nie zostały tam zamieszczone, źródła można pobrać ze strony domowej *http://www.goof.com/pcg/marc/fcrackzip.html*.

Program działa w linii komend, jednak jego składnia nie jest skomplikowana i ogranicza się do kilku parametrów określających jego zachowanie i metody szukania hasła. Wywołanie odbywa się przez wpisanie polecenia `fcrackzip`. Następnie określamy dodatkowe parametry. Chociaż program jest niewielki, oferuje bardzo zbliżone opcje do tych, które poznaliśmy na przykładzie *Johna the Rippera*. Na początek definiujemy sposób ataku, czyli wybieramy pomiędzy metodą *brute-force* a słownikową. Jeśli zdecydujemy się na pierwszą z nich, musimy do polecenia dodać parametr `-b`. Jeżeli natomiast wolimy metodę słownikową, ponieważ np. domyślamy się, że hasłem jest jakieś mało skomplikowane słowo, należy podać dodatkowy parametr `-D`. Teraz będziemy mogli określić ścieżkę do słownika, z którego mamy zamiar skorzystać. Słownik może być również plikiem przygotowanym przez użytkownika. Wymogiem programu jest, aby w każdej linii takiego pliku znajdowało się tylko jedno słowo oraz aby był on posortowany alfabetycznie.

W przypadku łamania hasła za pomocą słownika pole manewru jest dosyć ograniczone, ponieważ program posługuje się tylko gotową listą słów. Jak wspomnieliśmy już przy okazji omawiania programu John the Ripper, szukanie hasła metodą *brute-force* jest nie tylko obciążeniem dla procesora. Jeśli niedokładnie wybierzemy tryb pracy, tzn. źle określimy dodatkowe parametry zawężające obszar poszukiwań, proces deszyfracji stanie się strasznie czasochłonny. Ta sama sytuacja ma miejsce w przypadku *fcrackzipa*. Na początku należy określić wielkość liter. Jest to bardzo ważne, ponieważ wybór tylko dużych lub małych liter może radykalnie zawęzić krąg poszukiwań (pamiętajmy, że możemy wtedy złamać tylko hasła słabe lub o średniej sile). W kontekście czasu oczekiwania na wyniki, nakazanie programowi szukania zarówno dużych, jak i małych liter powinno być ostatecznością. Aby *fcrackzip* sprawdzał jedynie kombinacje składające się z małych liter, należy do polecenia `fcrackzip -b` dodać parametr `-c a`. Jeżeli zaś chcemy użyć jedynie dużych liter, dodajemy `-c A`. Czasem jednak może zdarzyć się, że będziemy musieli złamać hasło składające się z samych cyfr. Co wtedy zrobić? Sprawdzanie kombinacji literowych mija się z celem. Twórcy programu pomyśleli i o takiej ewentualności. Wystarczy, że dodamy parametr `-c 1`. Ostatnią ewentualnością są znaki dodatkowe, takie jak @, # czy $. Jeśli hasło składa się jedynie z tego typu znaków, to w zupełności wystarczy polecenie `fcrackzip -b -c !`. Biorąc pod uwagę skomplikowane hasła, trudno sobie wyobrazić sytuację, w której użytkownik nie mógłby łączyć poszczególnych parametrów. Załóżmy, że musimy złamać hasło składające się z małych liter, wśród których znajduje się cyfra. Dla utrudnienia właściciel hasła dodał znaki @ i #. Wygląda strasznie! Jak sobie z tym poradzić? W rzeczywistości nie stanowi to problemu – poza czasem, który komputer będzie potrzebował na odszyfrowanie takiego hasła. Wydajemy polecenie `fcrack -b -c a1:@#`. Na koniec określamy długość szukanego ciągu `-l długość`. I to wszystko, co możemy w tej chwili zrobić. Reszta zależy od wydajności komputera, samego programu i cierpliwości użytkownika.

Rysunek 4. *Strona internetowa programu ophcrack – dział Download*

PDF i hasło?

Jeden z najpopularniejszych i najbardziej przenośnych formatów ostatnich lat – PDF – coraz częściej jest zabezpieczany hasłem. Sytuacja jest podobna do wspomnianych wcześniej archiwów ZIP. Nawet jeżeli dokument został zabezpieczony hasłem, które ma uchronić zawartość przed nieautoryzowanym wyciekiem informacji, w rzeczywistości chroni go jedynie przed osobami mającymi małe doświadczenie z komputerami. Jest to pewien paradoks, ponieważ osoby naprawdę zainteresowane informacjami znajdującymi się w tak zabezpieczonym dokumencie PDF bez większych kłopotów mogą się do nich dostać. Istnieje kilka doskonałych programów, dzięki którym możemy w sposób podobny, jak w przypadku plików ZIP, ujawnić hasło. Jednym z najlepszych jest *pdfcrack*. Autorem jest Nacho Barrientos Arias. Pierwotnie program był pisany dla dystrybucji Debian, jednak obecnie może być używany także pod innymi dystrybucjami. Sposób użycia również nie budzi wątpliwości – nawet

użytkownicy, którzy nigdy nie interesowali się crackingiem, bez kłopotów są w stanie zdeszyfrować hasło.

Najbardziej zwięzła wersja polecenia ma postać `pdfcrack -f nazwa_pliku.pdf`. Jak możemy się spodziewać, w tym wypadku wyszukiwanie prawidłowego ciągu potrwa bardzo długo. Niezbędnym zabiegiem jest rozszerzenie tego polecenia o dodatkowe parametry. Niestety, dokumentacja na ten temat nie jest zbyt rozbudowana. Postaramy się jednak przedstawić opcje w taki sposób, aby ich obsługa nie sprawiała problemu. Powyżej wskazaliśmy już podstawową wersję polecenia uruchamiającego program. Naszym zadaniem będzie takie uzupełnienie go, aby wyszukiwanie trwało jak najkrócej. Na początek warto ustalić zakres znaków, które najprawdopodobniej są użyte w haśle. W przypadku *pdfcracka* jest troszeczkę więcej pracy niż przy poprzednich programach. Musimy tu bowiem sami wpisać każdy znak mogący wchodzić w zakres szukanego ciągu. Jeśli hasło składałoby się jedynie z cyfr, do polecenia bazowego należałoby dodać zawężenie zakresu w postaci `-c 0123456789`. Naturalnie definicja zakresu małych liter alfabetu przybierze postać `-c abcdefghijklmnopqrstuvwxyz`. Nie ma przeszkód, aby np. mieszać litery i cyfry. Oczywiście metoda brute-force nie jest jedyną obsługiwaną przez narzędzie. Jeśli ktoś ma gotowy słownik, nie ma przeszkód, aby z niego skorzystać. Wystarczy zasygnalizować to parametrem `-w słownik.txt`. Warto również określić minimalną i maksymalną długość szukanego ciągu. Do ustalenia pierwszej z tych wielkości służy opcja `-n`, po której podajemy wybraną wartość (np. `-n 3` stanowi, że próbowany ciąg ma być nie krótszy niż 3 znaki). Natomiast aby zdefiniować maksymalną długość, wpisujemy `-m` (oraz oczywiście odpowiednią wartość). Po ustawieniu wymienionych powyżej parametrów nie pozostaje nic innego jak wskazanie pliku zabezpieczonego hasłem i oczekiwanie na efekt pracy *pdfcracka*.

NTLM w pięć minut

Zastanawiasz się, czym jest NTLM? Jest to algorytm zabezpieczający systemy Windows XP oraz Windows 2003 Server. Chociaż od dawna funkcjonuje już Windows Vista, to jednak użytkownicy bardzo często korzystają właśnie z tych wcześniej wymienionych systemów. Windows XP jest aktualnie wykorzystywany w większości biur administracyjnych, szkół, bibliotek oraz we wszystkich innych miejscach, które wykupiły licencję publiczną. Czy oznacza to, że nie można włamać się do systemu Windows Vista? Nic podobnego. Jest to równie proste, jak w przypadku Windows XP, ale o tym poniżej. Wróćmy jednak do tematu. Podstawowe pytanie: *Co nas najbardziej interesuje w systemie?* Odpowiedź brzmi oczywiście: *Konto administratora*. Dostęp do tego konta jest najczęściej odpowiednio zabezpieczony (z oczywistych względów). W rzeczywistości jednak nawet skomplikowane hasła nie są barierą, której nie można pokonać. Aby uzyskać dostęp do takiego konta, niezbędne są trzy elementy: hash będący zakodowanym hasłem, odpowiedni program i czynnik najważniejszy... chwila nieuwagi gospodarza danego systemu.

Rozpocznijmy od najważniejszej kwestii, czyli pozyskania materiału do naszych badań nad zabezpieczeniami. Ogólnie rzecz biorąc, istnieje kilka sposobów na zdobycie najbardziej interesującego nas pliku, czyli SAM. Możemy go znaleźć w dwóch miejscach. Podstawowym katalogiem jest *C:\Windows\System32\conf*. Tu jednak z całą pewnością napotkamy na pewien problem, a mianowicie błąd braku dostępu. Drugim katalogiem, w którym najczęściej przechowywana jest kopia pliku SAM, jest *C:\Windows\repair*. A co zrobić, jeżeli nie ma kopii, a do oryginału nie można się dostać? Należy posłużyć się oprogramowaniem typu *pwdump4*. Dzięki temu narzędziu w prosty sposób pozyskamy odpowiednie hashe i jedyne, co nam pozostanie, to rozszyfrowanie ich. Przykładowe użycie tego programu może wyglądać tak: `pwdump4 /l /o:hasla.txt`. Pierwszy parametr określa, że przeszukiwany będzie komputer lokalny, natomiast drugi definiuje ścieżkę do pliku wynikowego. Jeżeli posiadamy już odpowiednie informacje, przekazujemy je do programu *ophcrack*. Teraz wystarczy uruchomić dekodowanie i czekać na gotowe hasła.

Niestety, *ophcrack* w wersji instalowanej na dysku twardym nie zawsze działa tak, jak powinien. Dużo wygodniej jest pobrać ze strony producenta (*http://ophcrack.sourceforge.net*) wersję LiveCD. Do wyboru mamy dwie opcje: *ophcrack* dedykowany dla XP lub Visty.

Wybieramy odpowiednią wersję, pobieramy plik ISO (aktualna wersja 2.0.1), a następnie wypalamy na płycie CD. Jak wskazuje nazwa, jest to wersja bootowalna, więc aby ją wczytać, należy ponownie uruchomić komputer i wybrać system z płyty. Jeśli wszystko przebiegnie bez kłopotów, pierwszym komunikatem, jaki powinniśmy ujrzeć, będzie informacja o katalogu, w którym prawdopodobnie znajduje się plik SAM. Uruchamiamy *ophcrack*, a kiedy dane zostaną wczytane, jedyne, co nam pozostaje – to wcisnąć przycisk *Crack*. Już po kilku minutach powinniśmy ujrzeć poprawne hasła dla poszczególnych użytkowników. Szybko, prosto i przyjemnie.

Podsumowanie

Jak widzimy, uzyskanie dostępu do danych, które w rzeczywistości są zabezpieczone jedynie pozornie, niekoniecznie wymaga niewyobrażalnych – jak zwykło się sądzić – umiejętności. Wystarczy odrobina pomysłowości, intuicji i przede wszystkim cierpliwości, natomiast resztę wykona odpowiednie oprogramowanie. Nawet tak pomysłowe rozwiązania, jak SHA-1 czy SHA-2, nie są w stanie zatrzymać osób, którym zależy na przejęciu kontroli nad danym komputerem (oczywiście o ile łamane hasło nie jest bardzo silne – jednak trudne do złamania hasła wykorzystuje, niestety, niewielu użytkowników). W artykule poruszyliśmy rozmaite formy pozyskiwania hasła oraz różne metody jego dekodowania. Nadrzędnym celem było pokazanie, że nie ma jednej, uniwersalnej metody na złamanie hasła. Czasami trzeba ciężko pracować na efekt finalny i niezaprzeczalnie należy uzbroić się w cierpliwość. Czy warto? Oceńcie sami...

Łukasz Ciesielski
Autor jest dziennikarzem, którego pasją stało się programowanie komputerowe oraz systemy z rodziny Linuxa (zwłaszcza Debian i Slackware). E-mail: lucas.ciesielski@gmail.com
Kontakt z autorem: *lucas.ciesielski@o2.pl*