Obrona

Słabe strony uwierzytelniania hasłami

Cezary G. Cerekwicki

stopień trudności

Większość istniejących systemów informatycznych używa uwierzytelniania przez hasła. Niniejszy tekst opisuje słabe punkty tego typu systemów, ale przedstawia też najlepsze znane obecnie rozwiązania, pozwalające je wzmocnić. Opisano problematykę ochrony systemu w najważniejszych przypadkach użycia uwierzytelniania poprzez hasła.

Hasła nieprzypadkowo są najpopularniejszym sposobem uwierzytelniania. Główną przyczyną znacznie mniejszej popularności innych technik (np. metod biometrycznych, różnego rodzaju kart itd.) jest ich dużo wyższa cena. Nawet w przypadku tzw. silnego uwierzytelnienia, a więc udowadniania swojej tożsamości na kilka sposobów, hasło (np. w postaci PIN-u albo haseł jednorazowych) jest z reguły jednym z nich.

Uproszczony nieco cykl życia hasła może wyglądać następująco:

- Hasło zostaje wymyślone przez użytkownika,
- hasło wędruje przez Sieć,
- hasło zostaje przetworzone przez system,
- hasło zostaje gdzieś zapisane ku pamięci,
- hasło zostaje zapisane przez program na maszynie klienckiej (np. przeglądarkę webową, klienta poczty),
- użytkownik loguje się,
- użytkownik zapomina hasła i korzysta z mechanizmu przypominania lub resetowania,
- hasło trafia do backupu systemowego,
- użytkownik otrzymuje pocztą listę haseł jednorazowych.

W konkretnej sytuacji może to wyglądać tak, jak w poniższym przykładzie. Użytkownik chce stworzyć sobie konto w portalu. W tym celu wypełnia formularz i podaje wymyślone przez siebie hasło. Hasło zostaje zapisane w jego Firefoksie, dodatkowo użytkownik zapisuje je sobie w pliku, w którym trzyma wszystkie swoje hasła. Wypełniony formularz podróżuje siecią do portalu, gdzie hasło jest przetwarzane. Użytkownik korzysta wielokrotnie z portalu, za każdym razem logując się.

Z artykułu dowiesz się

- gdzie kryją się słabości systemów uwierzytelniających hasłem,
- jak te słabości wyeliminować.

Co powinieneś wiedzieć

- wskazane jest rozumienie podstaw terminologii kryptologicznej i elementarna znajomość informatyki.

Tworzenie haseł

Idealny użytkownik powinien przestrzegać następujących reguł bezpieczeństwa:

- Używaj długich i skomplikowanych haseł,
- hasło nie powinno być słowem ani składać się ze słów,
- nie używaj tego samego hasła na różnych kontach,
- nie zapisuj nigdzie haseł,
- systematycznie zmieniaj hasła (z częstotliwością proporcjonalną do istotności chronionego zasobu),
- ujawnienie któregokolwiek z haseł nie powinno stanowić żadnej wskazówki co do pozostałych,
- ilekroć wystąpi podejrzenie, że dane hasło mogło zostać ujawnione, należy je jak najszybciej zmienić,
- nie pozwalaj żadnym programom na zapamiętywanie hasła,
- zawsze się wyloguj po skończeniu pracy,
- zawsze blokuj pulpit komputera, kiedy od niego na chwilę odchodzisz,
- jeśli nie zamierzasz już dłużej korzystać z danego komputera, należy oczyścić go ze wszystkich prywatnych danych, w szczególności wtedy, gdy są w nim zapisane hasła i inne poufne informacje.

Jednak idealni użytkownicy nie istnieją. Długość hasła można (i trzeba) wymusić, skomplikowaną strukturę hasła również, ale to już mniej ważne. Dlaczego? Ponieważ maksymalna ilość iteracji w ataku brutalnym to Sd, gdzie S to wielkość użytego słownika, a d to długość hasła. A więc gołym okiem widać, że każdy dodatkowy znak długości zwiększa przestrzeń haseł o rząd wielkości (przy typowych wielkościach S), a dodatkowy znak w słowniku o znacznie mniej. Wniosek: długość jest ważniejsza niż różnorodność znaków, zatem to przede wszystkim długość powinniśmy wymuszać na użytkownikach. Wiele programów wymusza systematyczne zmienianie hasła, w dodatku pamiętając stare hasła i nie pozwalając, by się powtórzyły. Niektórych ludzi to denerwuje, przez co szukają oni sposobu, jak oszukać taki program. Jednym z takich sposobów jest dopisywanie do swojego ulubionego hasła numeru aktualnego miesiąca. Zastanówmy się jednak nad tym, co się stanie, gdy włamywacz uzyska dostęp do haseł nieaktualnych? Widząc, że historia haseł danej osoby to *Ronaldo-4* oraz *Ronaldo-10*, wypróbuje inne cyferki w miejsce tych, które już zna. Widząc hasła michal i piotrek zapewne sprawdzi wszystkie imiona z kalendarza. Jeśli tylko istnieje możliwość wywnioskowania aktualnego hasła z haseł nieaktualnych, to ich zmienianie istotnie obniża poziom bezpieczeństwa (ale ciągle ma sens). Podany wyżej scenariusz można też potraktować jako przykład konfliktu bezpieczeństwa z użytecznością – systematyczne zmienianie haseł niewątpliwie jest korzystne, ale wiąże się z niemiłą dla użytkowników koniecznością ciągłego ich wymyślania i zapamiętywania. Można więc przypuszczać, że odbędzie się to ze szkodą dla ich jakości. Rodzi to pewne konsekwencje dla autorów oprogramowania. O ile reguły długości czy ilości użytych znaków łatwo na użytkownikach wymusić, o tyle zaprojektowanie algorytmu, który będzie w stanie stwierdzić, że jedne hasła można wywnioskować z innych, jest niemożliwe (wyjąwszy trywialne przypadki wnioskowania, np. dla haseł typu *Iza1*, *Iza2*). Żaden program nie stwierdzi, że hasła *YellowSubmarine* i *HeyJude* związane są z zespołem The Beatles, co dla włamywacza może być cenną i nietrudną do wywnioskowania wskazówką. Możliwość wnioskowania jest też zagrożeniem dla zasady tworzenia różnych haseł dla różnych kont. Możemy sobie wyobrazić scenariusz, w którym napastnik zdobywa hasło do słabo chronionego zasobu (np. prywatnego konta e-mail) i na jego podstawie zgaduje hasło do czegoś ważniejszego, np. banku internetowego. Tak więc kolejną pożądaną cechą systemu haseł jest brak możliwości wnioskowania: odkrycie jednego hasła nie powinno dawać napastnikowi żadnej użytecznej informacji co do pozostałych haseł (zwłaszcza haseł aktualnych oraz przyszłych). Innych ważnych reguł bezpieczeństwa również nie da się upilnować metodami technicznymi, dlatego ważna jest edukacja użytkowników. Istnieje prosty sposób na tworzenie dobrych haseł. Na początek potrzebujemy jakiegoś cytatu z literatury, tekstu piosenki, albo jakiegokolwiek innego źródła tekstu. Jako przykładem posłużę się pierwszymi wersami Pana Tadeusza Adama Mickiewicza:

Litwo, ojczyzno moja, ty jesteś jak zdrowie
Ile cię trzeba cenić, ten tylko się dowie, kto cię stracił

Druga potrzebna nam rzecz to algorytm przekształcenia takiego tekstu na hasło. Może on np. wyglądać tak:

- Początek wiersza zamieniamy na jego numer porządkowy i włączamy do hasła,
- koniec wiersza zamieniamy na znak / i włączamy do hasła,
- do hasła włączamy pierwszą literę każdego słowa,
- do hasła włączamy znaki interpunkcyjne,
- pozostałe znaki ignorujemy.

Stosując nasz algorytm, tworzymy następujące hasło:

„1L,om,tjjz/2Ictc,ttsd,kcs/”

Ma ono 26 znaków długości, zawiera małe i duże litery oraz jeden znak interpunkcyjny i jeden specjalny, a więc jest obecnie bardzo silne. Oczywiście lepiej nie posługiwać się podanym wyżej przykładem algorytmu, a już na pewno nie tekstu. Należy wybrać sobie wiele źródeł tekstu (mogą być to wybrane zwrotki piosenek, wierszy, cytaty znanych osobistości, nadruki na lekarstwach, dowcipy, przysłowia

itd.), aby móc tworzyć wiele różnych haseł.

Tekst, którego będziemy używać jako źródła hasła, powinien mieć następujące cechy:

- Być łatwy do zapamiętania (bardzo dokładnego),
- nikt nie powinien go zbyt łatwo kojarzyć z użytkownikiem,
- nie powinien być zbyt krótki.

Następnie należy opracować własną wersję algorytmu (opierając się na podanym przeze mnie przykładzie). Można na przykład wprowadzić regułę zamieniania spacji na dany znak (np. %), przecinki w tekście źródłowym zapisywać jako inne znaki (np. @) w haśle, używać ostatnich, a nie pierwszych liter każdego słowa itd.

Nasz algorytm powinien:

- Być łatwy do zapamiętania (bardzo dokładnego),
- generować hasła z elementami co najmniej trzech zbiorów (małe litery, duże litery, cyfry, znaki interpunkcyjne, znaki specjalne),
- generować długie hasła (co najmniej 10 znaków).

Oczywiście należy trzymać w tajemnicy hasła, algorytm oraz tekst źródłowy.

Transport haseł

Hasła mogą być transportowane w sieci w przypadku:

- Tworzenia konta,
- logowania się,
- zmieniania hasła.

Drugi przypadek użycia jest zdecydowanie najczęstszy, zatem przy jego projektowaniu trzeba zachować szczególną ostrożność. Podstawowym zagrożeniem jest fakt, że hasła podróżują przez sieci publiczne, gdzie mogą być czytane przez administratorów (legalnych i samozwańczych) każdej z tych sieci. Klasycznym rozwiązaniem tego problemu jest użycie SSL. Jest to rozwiązanie bardzo dobre, ale ma swoje minusy, w tym potencjalną podatność na atak man-in-the-middle, degradacje wersji protokołu itd. Rozwiązanie to jest też kosztowne, bo wymaga osobnego publicznego IP oraz wykupienia usługi firmy certyfikującej. Najlepiej (oprócz SSL) wprowadzić jeszcze jedną warstwę ochrony. Najlepszy tego typu mechanizm oferuje protokół *Secure Remote Password*, który realizuje dowód z wiedzą zerową (a więc nawet wtedy, gdy atakujący ma możliwość czytania i zmieniania wszystkich komunikatów pomiędzy serwerem i klientem, nie jest w stanie odzyskać hasła). Protokół SRP opisałem szczegółowo w osobnym artykule. Inne rozwiązania to np. używane przez niektóre banki podawanie tylko części hasła. To rozwiązanie jest w oczywisty sposób lepsze niż podawanie hasła w całości i w oczywisty sposób gorsze od SRP, gdzie hasło nigdy nie jest przesyłane siecią ani w całości, ani w części. Teoretycznie, jeśli ktoś podsłucha dostatecznie wiele sesji, będzie w stanie poskładać sobie całe hasło. Potencjalnie najbezpieczniejszym rozwiązaniem (nawet bezpieczniejszym niż SRP) jest użycie haseł jednorazowych. W tym przypadku nawet przejęcie hasła nic atakującemu nie daje, ponieważ ani nic tym hasłem nie uwierzytelni, ani nie będzie w stanie (w przypadku optymalnym, więcej na ten temat w dalszej części tekstu) wywnioskować z niego haseł przyszłych. Hasła jednorazowe mają jednak dość oczywisty minus – trzeba je bezpiecznie dostarczyć użytkownikowi. Najczęściej robi się to poprzez odpowiednio zabezpieczony list wysyłany pocztą albo przez SMS. Oba rozwiązania niestety kosztują. Hasła jednorazowe dostarczane przez SMS są bezpieczniejsze, ponieważ ich czas ważności jest niedługi. Napastnik musi skorzystać z hasła natychmiast po jego zdobyciu. W przypadku listy haseł jednorazowych jej ujawnienie jest dużo bardziej szkodliwe. Te hasła nie mają czasu ważności, a w dodatku ujawnieniu podlega cała lista haseł przyszłych, a nie tylko jedno. Jeszcze jednym podejściem do haseł jednorazowych są tokeny. Rozwiązanie to jest zdecydowanie bardzo bezpiecznym sposobem generowania haseł jednorazowych, ale też prawdopodobnie najdroższym. Token jest de facto specjalistycznym kalkulatorem kryptograficznym. Przechowuje pewnego rodzaju klucz prywatny (inny dla każdego klienta banku), ma również klawiaturę pozwalającą na wprowadzenie kodu. Wygenerowane przez niego hasło jednorazowe jest skrótem kryptograficznym klucza prywatnego oraz wprowadzonego kodu. Zauważmy, że uwierzytelnienie poprzez hasło jednorazowe wygenerowane przez token realizuje protokół z wiedzą zerową (którego bezpieczeństwo jest zależne tylko od nieodwracalności użytego algorytmu skrótu kryptograficznego).

Oceniając jakość rozwiązania zabezpieczającego transport haseł należy mieć na uwadze jego odporność na wszystkie znane metody ataku. Projektanci nieposiadający odpowiedniej wiedzy na ich temat mogą łatwo wpaść w jedną z licznych zasadzek kryjących się na tym polu. Na przykład naiwne metody szyfrowania haseł mogą być podatne na tzw. *replay attack*. Wyobraźmy sobie, że dany system zamiast przesyłać hasła w postaci tekstu jawnego, wysyła ich skróty SHA. Jeśli włamywacz podsłucha taką konwersację, nie będzie w stanie odczytać ze skrótu hasła w postaci jawnej, bo równałoby się to odwróceniu funkcji skrótu kryptograficznego. Ale takie odwrócenie wcale nie będzie mu potrzebne! Może bowiem sam nawiązać sesję i w miejscu, w którym serwer będzie oczekiwać skrótu SHA hasła włamywacz prześle mu przejęty wcześniej skrót. W ten sposób będzie się mógł poprawnie uwierzytelnić w ogóle nie znając hasła. Dlatego w każdym współczesnym protokole tak dużą wagę przywiązuje się do randomizacji przesyłanych przez sieć komu-

W Sieci

Na tej stronie znajduje się znakomita książka o pisaniu bezpiecznych programów:

- *http://www.dwheeler.com/secure-programs/*

Tu znajduje się kalkulator SRP:

- *http://srp.stanford.edu/demo/demo.html*

nikatów. Niedopuszczalna jest sytuacja, w której przesłany komunikat mógłby być w przyszłości przydatny dla potencjalnych podsłuchiwaczy. Pouczającym doświadczeniem jest zapoznanie się z kalkulatorem SRP, który pokazuje wszystkie wartości pośrednie używane w obliczeniach. Proponuję pobawić się przyciskami *randomize* i zwróceniem uwagi na to, które wartości się zmieniają i jak bardzo. SRP ma trzy wartości losowe i gdy zmieni się dowolna z nich (a dwie są losowane osobno dla każdej sesji, przy czym jedna jest losowana w sposób w pełni kontrolowany przez chroniony system, a więc można upilnować, by losowanie było kryptograficznie bezpieczne), zmieniają się wszystkie wymieniane przez publiczne łącze wrażliwe komunikaty. Co więcej, zasadnicze równania zapewniające uwierzytelnienie są ciągle prawdziwe, niezależnie od wylosowanych wartości!

Pułapek typu *replay attack* jest naturalnie o wiele więcej, dlatego każda osoba zamierzająca projektować systemy zapewniające wyso-

O autorze

Autor jest z wykształcenia informatykiem i politologiem. Pracował jako programista, administrator, konsultant, tłumacz, koordynator międzynarodowych projektów, dziennikarz i publicysta. Pisał programy w dziesięciu językach programowania (od asemblerów po języki skryptowe) w czterech systemach operacyjnych, na dwóch platformach sprzętowych.
Kontakt z autorem: *cerekwicki@tlen.pl*

ki poziom bezpieczeństwa musi być ich świadoma.

Przechowywanie haseł

System musi przechowywać jakąś informację, która później pozwoli mu na sprawdzenie, czy użytkownik podał właściwe hasło. Najprościej przechowywać samo hasło w postaci jawnej, ale naturalnie jest to podejście najgorsze z punktu widzenia bezpieczeństwa. Dostęp do wrażliwego pliku czy tabeli w bazie danych jest wprawdzie zabezpieczany klasycznymi metodami kontroli dostępu, ale poleganie wyłącznie na tej linii obrony od dawna jest uważane za niewystarczające.

Niezłym rozwiązaniem jest to, jakie stosowane jest przez Linuksa. Podczas tworzenia konta losowana jest liczba zwana solą. Sól nie musi być tajna, musi natomiast być gdzieś zapisana w systemie. Hasło jest konkatenowane (sklejane) z solą, a następnie liczony jest skrót kryptograficzny wyniku konkatenacji i to on jest zapisany w systemie. Jeśli włamywacz przejmie plik ze skrótami haseł, będzie musiał jeszcze przeprowadzić atak brutalny lub słownikowy, aby odzyskać hasło.

Użycie soli jest istotne, żeby zabezpieczyć się przed atakiem z obliczonymi wcześniej bazami skrótów kryptograficznych. W Internecie można znaleźć gotowe mapowania typowych haseł na ich skróty dla danej funkcji (najczęściej dla MD5). Dodanie soli czyni takie ataki nieefektywnymi (bo zwiększa to przestrzeń do przeszukania aż o kilka, kilkanaście rzędów wartości, w zależności od tego, ile bitów będzie miała sól). Efekt użycia soli polega też na tym, że nawet jeśli kilka kont będzie miało te same hasła, ich skróty kryptograficzne (czy też liczby weryfikujące) będą różne, zatem nie podpowie to niczego potencjalnemu włamywaczowi. Innym teoretycznym zagrożeniem jest atak przeciwdziedzinowy na użytą funkcję skrótu kryptograficznego, ale obecnie nawet

dla starej i wielokrotnie atakowanej funkcji MD5 nie udało się przeprowadzić skutecznie takiego ataku i szansa na odnalezienie tego typu podatności jest raczej niewielka, zatem trudno to zagrożenie traktować poważnie. Swoją drogą, gdyby któraś z powszechnie używanych funkcji skrótu (SHA, MD5) okazała się podatna na ataki przeciwdziedzinowe, oznaczałoby to automatyczną kompromitację olbrzymiej ilości systemów i de facto rewolucję w dziedzinie bezpieczeństwa informacyjnego. Zapewne najlepszym rozwiązaniem przechowywania haseł jest zastosowana w protokole *Secure Remote Password* (SRP) liczba weryfikująca. Liczba ta jest silnie zrandomizowana, a hasło chronione jest dodatkowo podwójną operacją liczenia skrótu kryptograficznego. Implementując kod przechowujący hasła, warto go dodatkowo zabezpieczyć przed atakami lokalnymi. Nowoczesne biblioteki programistyczne dostarczają do tego celu odpowiednich narzędzi, np. w .NET 2.0 mamy do dyspozycji klasę *SecureString*, przeznaczoną do przechowywania danych wrażliwych (haseł, PIN-ów, numerów kart kredytowych itd.) Klasa ta zapewnia automatyczne szyfrowanie swojej zawartości oraz umożliwia bezpieczne skasowanie. Przed jej użyciem należy uważnie zapoznać się z dokumentacją, aby nie popełnić jakiejś gafy (o to niestety łatwo podczas implementacji mechanizmów o podwyższonym standardzie bezpieczeństwa). Warto zapoznać się z podobną funkcjonalnością, jaką oferuje wybrane przez nas środowisko (mają ją zaimplementowaną wszystkie profesjonalne środowiska) lub, w przypadku braku tego typu wsparcia, spróbować je zaimplementować samodzielnie. Bezpieczeństwo można zwiększyć wieloma prostymi zabiegami, m.in. nadpisywaniem użytych zmiennych tymczasowych przed ich zwolnieniem (przynajmniej tych, w których były zapisane wrażliwe dane). Więcej informacji na ten temat można znaleźć w książce wymienionej w ramce *W sieci*.

Bezpieczeństwo maszyny klienckiej

Zapisywanie haseł w aplikacjach czy różnego rodzaju Menadżerach haseł rodzi potencjalną podatność. Jeśli z tych haseł może skorzystać legalny użytkownik, to może też i nielegalny. Co więcej, mechanizmy zabezpieczające składowane hasła często są bardzo słabe (np. wyciągnięcie haseł z popularnego komunikatora Gadu Gadu było wielokrotnie demonstrowane).

Inne zagrożenia to różnego rodzaju konie trojańskie, w szczególności keyloggery. Tu pomóc może jedynie edukacja użytkowników, systematyczne skany antywirusowe, dobry firewall i jakiś mechanizm ochrony integralności systemu (np. tripwire czy Kerio Personal Firewall). Skuteczne zarządzanie osobistymi firewallami albo mechanizmami kontroli integralności jest niebanalne i wymaga dużej i ciągle aktualizowanej wiedzy. To zdecydowanie najsłabszy punkt każdego systemu, zarządzany przez najmniej kompetentne osoby i najmniej kontrolowany.

Zauważmy, że nawet tutaj użycie haseł jednorazowych znacząco utrudnia włamywaczowi pracę. W typowym przypadku użycia hasło pojawi się w zasięgu keyloggera na kilkanaście, maksimum kilkadziesiąt sekund przed jego unieważnieniem. Tak więc okno czasowe dla potencjalnego ataku jest dość wąskie (jego dokładna wielkość zależy od czasu unieważnienia hasła jednorazowego od momentu zażądania go przez system). A w tym oknie należałoby zrobić kilka rzeczy: doprowadzić do zerwania sesji przez legalnego użytkownika, nawiązać własną nielegalną sesję i posłużyć się przejętym hasłem.

Dla projektantów systemów informatycznych płynie z tego oczywisty wniosek: należy bezwzględnie wprowadzić *timeout* dla haseł jednorazowych. Jeśli zażądane hasło zostanie wprowadzone po jego upłynięciu, nie powinno zostać przyjęte.

Hasła jednorazowe

Hasła jednorazowe można wygenerować na przynajmniej dwa sposoby. Najlepszy (ale też najtrudniejszy) sposób to sekwencja losowa (nie mylić z pseudolosową). Nieco mniej bezpieczny (ale łatwiejszy do implementacji) sposób to obliczenie *n* haseł na zasadzie *hasło(n) = sha(hasło(n-1))* i używanie ich w odwrotnej kolejności. Wówczas przewidzenie przyszłych haseł sprowadza się do przeprowadzenia ataku przeciwdziedzinowego na użytą funkcję skrótu. W przypadku sekwencji losowej nie grozi nam kryptoanaliza haseł wykorzystanych, ponieważ nie da się z nich wywnioskować haseł przyszłych. Wynika to z faktu, że w sekwencji losowej nie ma żadnych zależności między poszczególnymi jej elementami: są one całkowicie przypadkowe. Dlatego jest bardzo ważne, żeby nie posługiwać się sekwencją pseudolosową, która nie ma takiej własności.

Słabość tego podejścia polega na trudności w automatycznym generowaniu sekwencji naprawdę losowych. Zrealizowanie wydajnego generatora liczb losowych wymaga specjalnego sprzętu. Na zwykłym pececie można co najwyżej generować skończone ilości liczb losowych, zależnie od ilości możliwej do zebrania entropii. Nieźle robi to kernel Linuksa, pod warunkiem, że użytkownik dostarcza odpowiedniej ilości entropii poprzez stukanie w klawiaturę oraz ruszanie myszką.

Podsumowanie

W tekście opisano szereg zagadnień związanych z projektowaniem systemów informatycznych, w których uwierzytelnienie odbywa się poprzez podanie hasła. Wskazano miejsca szczególnie podatne na ataki oraz sposoby, na jakie można wzmocnić ich ochronę. ●