Protokoły z hasłem.

Część II

Stefan Dziembowski

Jak przechowywać hasła użytkowników na serwerze?

Zamiast przechowywać je otwartym tekstem lepiej przechowywać ich **hash**.

Czyli w jakimś pliku trzymamy

$$L := ((\mathrm{ID}_1, H(\pi_1)), \ldots, (\mathrm{ID}_n, H(\pi_n)))$$

(gdzie H jest ustaloną funkcją hashującą, a ID_i jest identyfikatorem użytkownika.)

Jak użytkownik ID_i się loguje, to pytamy go o hasło π i sprawdzamy, czy $H(\pi) = H(\pi_i)$.

Historia

Dawno temu uważano, że L może być ujawnione, bo nie zdradza informacji na temat $\pi_1, \ldots, \pi_n$. Dlatego w starych uniksach trzymano je w pliku `/etc/passwd`.

Obecnie nie jest to uważane za bezpieczne, bo możliwy jest atak słownikowy.

Jak wygląda atak słownikowy?

$$L := ((\mathrm{ID}_1, H(\pi_1)), \ldots, (\mathrm{ID}_n, H(\pi_n)))$$

Dla każdego $\tilde{\pi}$ ze słownika sprawdzamy, czy

$$H(\tilde{\pi}) \in \{H(\pi_1), \ldots, H(\pi_n)\}.$$

Jeśli znajdziemy takie $\tilde{\pi}$ i i, że

$$H(\tilde{\pi}) = H(\pi_i)$$

to wygraliśmy!

Jak się trochę zabezpieczyć

Aby utrudnić powyższy atak stosuje się **solenie** haseł.

Solenie haseł

Dla każdego użytkownika ID_i jednorazowo losujemy **sól** s_i.

W pliku haseł trzymamy $(s_i, H(\pi, s_i))$

(Aby zweryfikować hasło $\tilde{\pi}$ użytkownika ID_i sprawdzamy, czy $H(\tilde{\pi}, s_i) = H(\pi, s_i)$).

To przynajmniej trochę spowalnia atak.

Wzmacnianie haseł [1]

Można spowolnić obliczanie H.

Pomysł [AMLN]

Nie ujawniamy soli. Weryfikacja wymaga sprawdzenia wszystkich możliwych wartości soli.

Jeśli np. sól jest losowym ciągiem bitów długości 11, to w ten sposób

- spowolnimy weryfikację hasła średnio $2^{10} (= 1024)$ razy (to często nie boli).
- spowolnimy atak słownikowy tyle samo razy (to może przeciwnika zaboleć).

Wzmacnianie haseł [2]

Inny pomysł:

Aplikujemy funkcję H wielokrotnie

Jeśli H' jest funkcją hashującą, to definiujemy H jako

$$H(x) = \underbrace{H' \circ \cdots \circ H'}_{t \text{ razy}}(x)$$

(gdzie t jest parametrem).

Powinno zadziałać podobnie.

Hasła w Uniksie (`crypt()`)

Kiedyś to wyglądało tak (dokładny opis znajduje się w [MvOV01], s. 393):

Funkcją jednokierunkową H' jest trochę zmodyfikowany DES: DES*: $H'(x) := \text{DES}^*(x, (0, \ldots, 0))$ (x podajemy DESowi jako klucz).

Co to jest DES*?

Jest to DES lekko zmodyfikowany za pomocą soli.

Dlaczego tak robimy?

Żeby utrudnić ataki słownikowe za pomocą hardware'owych implementacji DESa (dostępnych za niewielkie pieniądze).

Wada tego rozwiązania?

DES ma klucz długości 7 bajtów (resztę `crypt()` obcina).

Dlatego w nowszych Uniksach funkcja `crypt` używa MD5.

Wada haseł wielorazowych

Kradzież hasła wielorazowego umożliwa poszycie się pod użytkownika. Taka kradzież w obecnych warunkach jest łatwa do przeprowadzenia:

- Jeśli logujemy się np. z kawiarni internetowej, to przechwycenie hasła (przez administratora kawiarni) jest trywialne. Może to zrobić za pomocą programu typu **keylogger**.
- Jeśli logujemy się z domu, to niby jest trochę bezpieczniej, ale zagrożeniem są
  - wirusy, robaki, etc.
  - zainstalowane programy (nikt nie wie, czy taki program nie zawiera keyloggera).

Zauważmy, że np. przechowywanie dodatkowego hasła na dysku niewiele zmienia.

Rozwiązanie

- Hasła jednorazowe — przechowywane np. w formie wydruku albo (to lepsze) na karcie ze „zdrapkami”.
- Tokeny typu RSA SecurID.
- Urządzenia które posiadają klawiaturę i wyświetlacz.
- Karty chipowe. To rozwiązanie ma wady:
  - Wymaga by terminale miały czytniki takich kart.
  - Skąd mamy wiedzieć jak dokładnie wygląda interakcja między kartą i komputerem?

oraz zaletę:

- Mało obciąża użytkownika.

System S/Key [1]

System S/Key jest systemem haseł jednorazowych [Hal94].

Przygotowanie listy haseł $p_1, \ldots, p_n$

s — losowy sekret
$p_n := s$
$p_{n-1} = H(p_n)$
$\vdots$
$p_0 = H(p_1)$.

$$p_0 \overset{H}{\leftarrow} p_1 \overset{H}{\leftarrow} p_2 \cdots \overset{H}{\leftarrow} p_n := s$$

Serwer przechowuje p_0

System S/Key [2]

$$p_0 \overset{H}{\leftarrow} p_1 \overset{H}{\leftarrow} p_2 \cdots \overset{H}{\leftarrow} p_n := s$$

Niezmiennik

Po i-tej rundzie serwer zna p_i.

Logowanie

Aby się zalogować w $i+1$-tej rundzie użytkownik przedstawia p_{i+1}.
Serwer sprawdza czy $H(p_{i+1}) = p_i$.

Zalety

Szybkość generowania haseł

Może to być zrobione nawet na prymitywnym sprzęcie.

Małe obciążenie pamięci serwera

Wystarczy, że serwer pamięta „ostatnie p_i”.

Model

Interesuje nas następujący scenariusz

Uczestnicy:

klient C i serwer S.

Klient zna klucz publiczny serwera (albo przynajmniej może go zweryfikować za pomocą certyfikatów).

Klient autentykuje się serwerowi za pomocą dzielonego klucza K_{AB}.

Ten scenariusz jest bardzo często spotykany w praktyce (banki, konta pocztowe, etc.). W tym przypadku klient z reguły korzysta z przeglądarki internetowej.

Metody rozwiązania

Dwa podejścia:

„praktyczne” Podejście praktyczne wykorzystuje to, że najczęściej klient dysponuje np. przeglądarką internetową z wbudowanym SSLem i wygodnie jest użyć tego mechanizmu.

„teoretyczne” W tym podejściu tworzymy protokół „od zera”.

Protokół Haleviego i Krawczyka

L — parametr, S — klucz publiczny serwera, π — hasło

Klient C		Serwer S
		$r_S \in_R \{0,1\}^L$
zalosuj K_0	$\xleftarrow{r_S}$	
	$\xrightarrow{E_S(K_0,\pi,r_S,C,S)}$	Odszyfruj i sprawdź
		$y := \mathrm{MAC}_{K_0}(r_S, S, C)$
	$\xleftarrow{y}$	$K_{CS} := \mathrm{MAC}_{K_0}(y)$
sprawdź y		
$K_{CS} = \mathrm{MAC}_{K_0}(y)$		

Jakie własności powinna mieć funkcja E_S

Prypomnijmy: klient wysyła $E_S(K_0, \pi, r_S, C, S)$.

Oczywiście nikt tego nie potrafi odszyfrować, bo S zna tylko serwer.

Nikt też nie potrafi takiej wiadomości sam wyprodukować, bo nie zna π.

Czy to wystarczy?

Nie!

Co jeśli na podstawie $E_S(K_0, \pi, r_S, C, S)$ ktoś potrafi wyprodukować $E_S(K_0', \pi, r_S, C, S)$ dla jakiegoś $K_0' \neq K_0$?

Problem

Zadanie przeciwnika (abstrakcyjnie)

Przeciwnik chciałby na postawie $E_S(M)$ wyprodukować:

$E_S(M')$ takie, że M' jest „jakoś związane” z M (ale nie jest równe M).

Czy może mu się udać?

Czy jest możliwe, że szyfrowanie zapewnia tajność, ale nie jest odporne na powyższy atak?

Przykład?

Non-malleability

Definicja (nieformalnie)

Kryptosystem jest **niepodatny** (ang.: *non-malleable*) jeśli na podstawie kryptogramu $C = E(M)$ przeciwnik nie jest w stanie obliczyć kryptogramu C' wiadomości M' („związanej z M”).

O większości kryptosystemów używanych w praktyce uważa się, że są niepodatne.

Wielu praktyków ignoruje tę kwestię.

Podejście praktyczne

1. Klient połączył sie za pomocą przeglądarki ze stroną banku `www.bank.pl`.
2. `www.bank.pl` dysponował odpowiednim certyfikatem, więc przeglądarka klienta potwierdziła tożsamość `www.bank.pl`.
3. Klient nie dysponuje żadnym kluczem publicznym, więc za pomocą SSL/TLS został osiągnięty kanał Γ: Klient — `www.bank.pl`, taki, że
   - Klient wie, że cokolwiek wyśle przez Γ, to będzie odczytane tylko przez `www.bank.pl`.
   - Klient wie, że cokolwiek przyszło przerz Γ, to było nadane przez `www.bank.pl`.

   Natomiast `www.bank.pl` „nic nie wie”.

Autentykacja klienta

Teraz powinien autentykowac się klient.

Poniższe kroki wykonywane są z użyciem Γ:

1. Klient zgłasza się z hasłem π do `www.bank.pl`.
2. `www.bank.pl` weryfikuje hasło π.
   Jeśli hasło się zgadza, to `www.bank.pl` wysyła do klienta **żeton autentykujący** (albo: **autentykator**) A.
   Z reguły A ma określony termin ważności.
3. Za każdym razem kiedy klient zgłasza się do `www.bank.pl`, to do tekstu komunikatu dodaje A.

Na czym opiera się bezpieczeństwo?

- Autentykator A musi być wybrany w taki sposób, żeby trudno było go zgadnąć.

  Może to być losowa liczba (ale wtedy trzeba za każdym razem patrzeć do bazy danych)

  Inny wariant: A jest obliczane jako MAC_K na jakichś danych związanych z klientem (K jest kluczem serwera).
- Autentykator powinien być wysyłany wyłącznie po SSLu (czyli kanałem Γ).

W jaki sposób klient przekazuje *A*

Do każdego późniejszego żądania HTTP klient musi dodać autentykator *A*.

W jaki sposób?

1. W URLu:
   `https://usosweb.mimuw.edu.pl/logowanie/konto.php`
   `id=100c597213d42e52070137441105ff58&_msg=1`
2. Za pomocą ciasteczka.
3. Za pomocą „hidden form fields”.

Autentykator w URLu

Na co trzeba uważać?

Nagłówek „Referer”.

O co chodzi?

Jeśli na stronie `http://www.poczta.pl/324wedsafase` kliknę na link `www.przestepcy.pl`, to:

serwer `www.przestepcy.pl` otrzyma informację, że na ich stronę przychodzę ze strony `http://www.poczta.pl/324wedsafase`.

Autentykator w ciasteczku

Co to są ciasteczka?

Są to informacje które dany URL może „nagrać” na przeglądarce..
Przy każdym łączeniu się z URLem przeglądarka wysyła wszystkie „ciasteczka” które dany URL na niej nagrał.

Na co trzeba uważać?

1. Ciasteczka powinny mieć ustawioną flagę „SSL Only”.
2. Ostrożnie z zapisywaniem na dysku (bodobno kiedyś można było całkiem sporo takich ciasteczek „wygooglować”.)

Analiza praktycznych systemów

Omówimy teraz część pracę

Kevin Fu, Emil Sit, Kendra Smith, and Nick Feamster
Dos and Don'ts of Client Authentication on the Web In the Proceedings of the 10th USENIX Security Symposium, Washington, D.C., August 2001.
W tej pracy dokonano analizy bezpieczeństwa wybranej grupy systemów autentykacji za pomocą hasła.

Klasyfikacja przeciwników

- **badacz** („interrogative”) — korzysta tylko z dostępnej publicznie wiedzy, może (w legalny sposób) wchodzić w interakcję z atakowany systemem (zakładać konta, próbować logować się, etc.).
  Odporność na taki tego typu to wymaganie minimalne!!!
- **podsłuchujący** — dodatkowo może podsłuchiwać całą komunikację między serwerem i klientami.
- **aktywny** — może aktywnie ingerować w komunikację między uczciwymi użytkownikami i serwerem.

Rezultat pracy

Z 27 analizowanych serwerów autorom udało się:

- włamać na 9
- w tym na 1 z nich udało się nawet otrzymać hasło użytkownika.

(głównie za pomocą ataków „badawczych”).

Konkretne przykłady

www.ichat.com

Ciasteczko jest xor-em identyfiaktora z ustalonym kluczem

username	fubob
password	aaaaaaaa
cookie	ichat cookie=0F160A0E1656233E21254D...
username	fub**b**b
password	aaaaaaaa
cookie	ichat cookie=0F160A0**3**1656233E21254D...

Konkretne przykłady - cd

New England Bride `nebride.com`

Ciasteczko jest równe identyfikatorowi użytkownika.

„We notified New England bride, but we are not sure if they understand the problem.”

`Fatbrain.com`

Autentykator (przekazywany w URLu) jest kolejną wartością globalnego licznika. Przeciwnik, który uzyska dostęp do konta może:

- zmienić adres alektroniczy klienta
- poprosić o przesłanie hasła na ten adres

Morał: takie rzeczy powinny by pod szczególną ochroną!

Konkretne przykłady - cd

`SprintPCS.com`

System używa ciasteczek.

Flaga „SSL Only” nie jest ustawiona, w związku z czym:

jeśli klient połączy się z niezabepieczoną częścią `SprintPCS.com`, to przeciwnik może podsłuchać ciasteczko.

Jeszcze jeden przykład – strona `WSJ.com`

Autorzy przedstawiają atak „badawczy” na stronę **Wall Street Journal**.

Atak ten pozwala na otrzymanie pełnego dostępu dowolnego konta w `WSJ.com`, włącznie z możliwością robienia zakupów na koszt właściciela konta.

Jak powstaje autentykator w `WSJ.com`?

Użyta jest uniksowa funkcja `crypt()` (w DES-owej wersji).

Kokretnie:

$$A := \texttt{crypt}(\text{UserName} \cdot \text{„March20”}).$$

Na czym polega problem

- `crypt`() patrzy tylko na pierwszych 8 znaków, więc wszystkie identyfikatory powyżej 8 znaków będą miały taki sam autentykator A.
- Nawet jakby `crypt`() było lepsze, to „tajny klucz” serwera („March20”) poddaje się atakowi słownikowemu.

Na dodatek:

- nie ma mechanizmu „rewokacji” ciasteczka (nawet jak się zmieni hasło, to przestępca ma cały czas dostęp do konta).
- Ciasteczko trwa „wiecznie” (niby ma ustawioną datę ważności, ale zadanie jej weryfikcji spoczywa na kliencie).
- Serwer akceptuje też ciasteczka pochodzące od nieistniejących użytkowników.

Przykłady z polskiego Internetu

Podamy teraz kilka przykładów z polskiego Internetu.

Nie będzie to analiza bezpieczeństwa, tylko krótki rzut oka na to jakie rozwiązania zastosowano.

www.lukas.com.pl

- Do autentykacji klient używa:
  - identyfikatora
  - hasła (które pamięta)
  - wartości z tokena RSA SecurID.

  Przy dokonywaniu przelewy klient musi ponownie wprowadzić hasło i (nową wartość) z tokena.
- Autentykator jest przesyłany za pomocą ciasteczka:

```
    Name:   rsa-local
 Content:   xyzaaa1ASD1J8324DU1394HDDQW9J91342R8CJ93S
            123ENIUD34D134ND134D1V90WRT90VIN28NF2U35X
            41JD1D4UF82=34HV29HF23498R34RDC234R314R13
    Host:   e-bank.lukas.com.pl
    Path:   /
Send For:   Encrypted connections only
 Expires:   at end of session
```

www.inteligo.pl

- Do autentykacji klient używa:
  - identyfikatora
  - hasła które pamięta
  - kodu z listy haseł jednorazowych (tylko do przelewów).
- Autentykator jest przesyłany za pomocą „hidden form„:

```
<input name="sd" type="hidden" value="
eqfc3214cj:92193E/W4Jsdfasu23hbbyubD:45v
mocsd9090JJIAJ02223121eosAfrfer2345nfrej
fe87234crnHF3W498RH2nfi2p34f234FB082347B
F324F1203123:1923dqeawdxASDASD3EXC2/WSwe">
```

www.gazeta.pl [1]

Do autentykacji klient używa:

- identyfikatora
- hasła które pamięta

Przy logowaniu się klient może zaznaczyć pole „Używaj bezpiecznej poczty”.

Jeśli ja zaznaczy, to skrypt Java dba o to, żeby hasło i login były przekazane za pomocą SSLa (do `ssl.gazeta.pl`).

Czy to rozwiązanie jest w pełni bezpieczne?

www.gazeta.pl [2]

Autentykator jest chyba przekazywany w URLu:

`https://ssl.gazeta.pl/poczta/0,46200.html?t=1115799`

(dodatkowo też w hidden form).

Oprócz tego serwer `www.gazeta.pl` wysyła zawsze (niezabepieczone) ciasteczko `JSESSIONID_GW` identyfikujące sesję, np:

```
CBLkZgvYdwBcQ3jwT6kqv10CGf7v1J2Z
JspvsXpRcftPkXkgpmZc!-1009905861
```

Autentykator z `www.gazeta.pl`

Autentykator zmienia się na w czasie:

`https://ssl.gazeta.pl/poczta/0,46200.html?t=1115799`

Po chwili:

`https://ssl.gazeta.pl/poczta/0,46200.html?t=1115800`

Autentykatory z dwóch różnych kont użyte w odstępie sekundy:

`1115802650954`

`1115802651884`

Można próbować kombinować :)

SSH

Innym protokołem, który dopuszcza (między innymi) autentykację przez hasło jest SSH.

SSH ma dwie wersje: SSH1 i SSH2.

Schemat działania jest podoby do protokołów opartych na SSL:

1. Ustanawiany jest kanał Γ

   Klient — Serwer,

   w którym autentykowany jest tylko serwer.

Jak autentykowany jest serwer?

Serwer autentykowany jest za pomocą metod kryptografii klucza publicznego.

Jeśli nie ma PKI, to oczywiście mam stary problem:

jak poznać klucz publiczny serwera.

Rozwiązanie w SSH

Utrzymujemy bazę znanych nam kluczy serwerów (np. `~/.ssh/.knownhosts`).

Jak się łączymy za pierwszym razem, to serwer wyświetla pytanie, czy dodać klucz do bazy:

```
The authenticity of host 'duch (10.1.3.2)' can't be established.
RSA key fingerprint is 17:49:d0:cd:55:01:e9:0b:4f:26:1d:75:c8:1e:40:64
Are you sure you want to continue connecting (yes/no)?
```

Co jeśli klucz się zmienił?

System odmawia połączenia:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle att
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
17:49:d0:cd:55:01:e9:0b:4f:26:1d:75:c8:1e:40:64.
Please contact your system administrator.
Add correct host key in /home/staff/std/.ssh/known_hosts to get rid of
Offending key in /home/staff/std/.ssh/known_hosts:21
RSA host key for duch has changed and you have requested strict checki
Host key verification failed.
```

Aby to usunąć trzeba przynajmniej w minimalnym stopniu wykazać się znajomością zasad działania programu.

Jak autentykowany jest klient?

- Za pomocą klucza publicznego

	SSH2	SSH1
	DSA, RSA	tylko RSA

- Za pomocą Kerberosa (tylko SSH1)
- Host-based (różne metody, w zależnośći od wersji 1 albo 2). Ta metoda umożliwia logowanie z jednej maszyny na drugą bez konieczności autentykacji.
  Wtedy za to maszyny powinny móc się wzajemnie autentykować.
  (W SSH1 tego wymagania nie było, co uważane było za rozwiązanie niebiezpieczne).
- Hasła
- Istnieją rozwiązania umożliwiające użycie haseł jednorazowych, albo tokenów.

Literatura I


M. Abadi, T. Mark, A. Lomas, and R. Needham.
Strengthening passwords.
SRC Technical Note 1997-033 (September/December 1997).


Neil M. Haller.
The s/key one-time password system.
In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, Feb 1994.


Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone.
Handbook of Applied Cryptography.
CRC Press, 2001.