Adatbiztonság ZH Név:
2015. május 7 Neptun kód:

1. a) Igaz-e, hogy az RSA rejtjelezés nem biztonságos olyan támadással szemben, amelyben a támadó azt tűzi ki, hogy megfejt egy y rejtett RSA blokkot úgy, hogy mindössze egy kérést küldhet egy dekódoló orákulumhoz, amely orákulum egy tetszőleges, $y^* \neq y$ blokkot dekódol a számára. **(3 p)**
b) Definiálja az existential forgery támadás digitális aláírás ellen? **(2 p)**
c) Igaz-e, hogy existential forgery támadást tudunk végrehajtani egyszerű RSA aláírás ellen anélkül hogy orákulum kérést küldenénk az aláíró algoritmusnak? (egy blokkos üzenet RSA aláírásában gondolkodjon) **(3 p)**

2. a) Definiálja a második őskép ellenállást hash függvények esetén? Mutasson támadási scenario-t arra az esetre, ha nem teljesülne ez a követelmény. **(3 p)**
b) Igaz-e, hogy nem ütközésellenálló kompressziós függvényt használó iterációs hash függvény lehet ütközésellenálló (kezdőérték ismert, fix)? **(3 p)**
c) Részletezzen egy születésnapi paradoxonon alapuló támadást hash függvény ellen? (n bites hash érték eseten jelezze a támadás algoritmikus komplexitását is) **(3 p)**

3.
a.) Milyen típusú sérülékenységet tartalmaz az alábbi forráskód és miért? **(2 pont)**

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
  char buffer[64];
  gets(buffer);
}
```

b.) A main függvényhez tartozó frame pointerhez (pl.: EBP) képest, hol helyezkedik el a buffer nevű lokális változó (mi a kezdőcíme)? Az egyszerűség kedvéért tekintsünk egy hagyományos x86-os architektúrát, és egy jól ismert compilert (pl.: gcc). **(3 pont)**
c.) Hogyan képes egy támadó kihasználni az alábbi sérülékenységet? Milyen címet írna felül a main függvényhez tartozó stackframe-en? Egy ilyen cím felülírásához milyen hosszú (bájtban mérve) input-ra lenne szüksége? **(7 pont)**

4. Adatbázisban tárolt jelszavakat próbálunk megszerezni blind SQL-injection támadás segítségével. Kérdésenként csak egy igen-nem döntésre van lehetőségünk. A támadó optimalizálja a támadást és a lehető legkevesebb lekérdezéssel meg tudja oldani a feladatot. Adja meg a szükséges SQL lekérdezések számát a következő esetekre:
a) a jelszó 8 karakteres angol abc (26 karakter) kis és nagybetűit és számokat tartalmaz és cleartext módon tárolt **(3 p)**

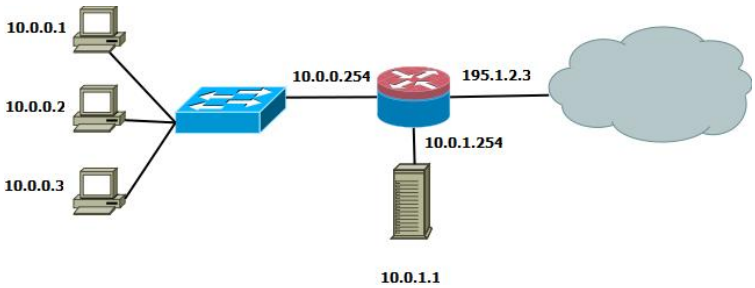
b) a jelszó egyszerű MD5 hashként van tárolva hexadecimális sztring formájában (output 128 bit) **(3 p)**

c) a jelszó 6 formátumban van tárolva, azaz: SHA-512-es hash (output:512 bit) 5000 szeres iterált számításával állítjuk elő, a salt 8 byte (a Base64 kódolás 64 betűs ABC-je szerint), a kapott hash Base64 kódolva van (86 karakter). A teljes tárolt sztring: 6<salt>$<base64 string>

Például:
6saltsalt$i0oEEeA3MlPQ3g8wZMe8ApDlh9f9F2MJhmcLkoigCTQ6RGwo.GvS3mKDJZqZK7OHamXmRPTkPQ86nYDw0AUXP1 **(3 p)**

A cél a hash-elt esetekben csupán a brute force jelszótöréshez szükséges adat megszerzése. Minden esetben a tárolt adatok 8 bites ASCII sztringek.

5. Tűzfal feladat



Az ábrán látható elrendezésben szeretnénk a Linux alapú tűzfalat bekonfigurálni. A belső hálózaton a gépek a 10.0.0.0/24 hálózatban vannak. A DMZ-ben (10.0.1.0/24) egy webszerver működik a 10.0.1.1-es címen. A tűzfal lábai a következők: eth0 kifelé, eth1 befelé, eth2 a DMZ irányába. Írjon iptables parancsokat a következő formátumban a részfeladatok megoldásához (a paraméterek sorrendjét lehetőleg ne változtassa meg, az alapértelmezett szabályokra ne alapozzon):

```
iptables [-t TÁBLANÉV] -A LÁNC [-p PROTOCOL] [-i INIF] [-o OUTIF] [-s SOURCE] [--sport SPORT] [-d DESTINATION] [--dport DPORT][--to ADD:PORT] -j ACTION
```

a) A belső hálózaton lévő gépek elérhetik a webszerver HTTP portját a DMZ-ben, és az válaszolhat is, ha az interfészek és a címek megfelelőek (állapotmentes megoldást írjon, 2 parancs) (**2 p**).

b) A tűzfalon futó SSH szerver csak a belső hálózatból kaphasson csomagot, és a tűzfal mint feladó általában is csak a belső hálózatnak küldhessen csomagot (ügyeljen a megfelelő lánc választásra, az interfészeket nem kell megadni, 4 parancs) (**4 p**).

c) A webszerver a külső hálózatból is elérhető legyen a nyilvános cím megfelelő portjain (HTTP és HTTPS forgalom is lehetséges legyen, 2 parancs) (**4 p**)

d) A tűzfalon áthaladó bármilyen LDAP-nak címzett forgalom (389-es port) logolva legyen (1 parancs). (**1 p**)

e) Sorolja fel, hogy ingress szűrés esetén milyen szabályokra lenne szükség (elég mondatban, nem kell szabály, 2 mondat). (**1 p**)

6. Tekintsük az alábbi /etc/passwd file részletet:

```
u1:x:1003:1004:,,,:/home/u1:/bin/bash
u2:x:1004:1005:,,,:/home/u2:/bin/bash
u3:x:1005:1006:,,,:/home/u3:/bin/bash
u4:x:1006:1007:,,,:/home/u4:/bin/bash
```

Az /etc/group file releváns része:

```
u1:x:1004:
u2:x:1005:
u3:x:1006:
u4:x:1007:
g1:x:1008:u1,u2
g2:x:1009:u2,u3,u4
g3:x:1010:u2,u3
```

A fájl hozzáférési jogosultságok az alábbiak:

```
root@gotcha:/adatbizt# ls -la
total 16
drwxr-xr-x  4 root root 4096 2011-04-22 10:49 .
drwxr-xr-x 25 root root 4096 2011-04-22 10:51 ..
drwxrwsr-x  2 u1   g1   4096 2011-04-22 10:50 d1
drwxr-xr--  2 u2   g1   4096 2011-04-22 10:50 d2

root@gotcha:/adatbizt# ls -la d1
total 20
drwxrwsr-x 2 u1   g1   4096 2011-04-22 10:50 .
drwxr-xr-x 4 root root 4096 2011-04-22 10:49 ..
-rw------- 1 u1   root    4 2011-04-22 10:50 f1
-rw-rw-r-- 1 u1   g1     16 2011-04-22 10:50 f2
-rwxrwxrwx 1 u1   g2      8 2011-04-22 10:50 f3

root@gotcha:/adatbizt# ls -la d2
total 16
drwxr-xr-- 2 u2   g1   4096 2011-04-22 10:50 .
drwxr-xr-x 4 root root 4096 2011-04-22 10:49 ..
-rw-r--r-- 1 root g1      7 2011-04-22 10:50 f4
--w------- 1 root g1      6 2011-04-22 10:50 f5
```

a.) mely felhasználók tudják kitörölni a d1/f1 fájlt és miért? (rm d1/f1) **(2 p)**
b.) mely felhasználóknál fut le sikeresen a cp d2/f4 d1/f6 parancs? **(2 p)**
c.) az u1 felhasználó (u1 aktív csoporttal) készít egy új fájlt d1-ben (touch d1/fu1), milyen csoport lesz a tulajdonosa a létrejövő fájlnak? **(2 p)**
d.) a root felasználó mely fájlokat tudja törölni az f1 alkönyvtárban? **(2 p)**
e.) ki tudja végrehajtani sikeresen az f2 fájl olvasási jogának teljes törlését? (chmod a-r d1/f2) **(2 p)**

Pontozás: 1: 0-24, 2: 25-33, 3: 34-42, 4: 43-51, 5: 52-60

	a	b	c	d	e	
1						
2						
3						
4						
5						
6						

Adatbiztonság ZH megoldások
2015. május 7

1.
a) Igaz. Vak-aláírás ötletét alkalmazza a támadó, $y^*=R^e y \pmod m$ blokkra kér dekódolást az orákulumtól, ahol R egy véletlen, mod m invertálható elem, majd mod m osztással eliminálja R elemet.
c) Igaz. Válasszuk tetszőlegesen egy x blokkot. Az $y=x^e \pmod m$ "üzenet" aláírása x.

2.
b) Tk. 4.2. feladat. Nem. Könnyű feladat m1, m2 egy blokk hosszú ütköző blokkot találni. Ezek felhasználásával, az iterálás miatt M1=[m1,m], M2=[m2,m] dokumentumok is ütközők lesznek, tetszőleges m esetén.

3.
a.) Milyen típusú sérülékenységet tartalmaz az alábbi forráskód és miért?

[+] Stack overflow/buffer overflow (1p)
[+] A buffer lokális változóba a gets() függvény segítségével tetszőleges méretű felhasználói inputot lehet elhelyezni, s így a bufferből kiindexelve érvénytelen címre mutató pointert tud egy esetleges támadó dereferálni. (1p)

b.) A main függvényhez tartozó frame pointerhez (pl.: EBP) képest, hol helyezkedik el a buffer nevű lokális változó (mi a kezdőcíme)? Az egyszerűség kedvéért tekintsünk egy hagyományos x86-os architektúrát, és egy jól ismert compilert (pl.: gcc).

[+] EBP-64, hiszen egy 64 bájtos lokális változóról van szó, s alignment nélkül az EBP alatt (alacsonyobb címen) közvetlenül helyezkedik el. (3p)

c.) Hogyan képes egy támadó kihasználni az alábbi sérülékenységet? Milyen címet írna felül a main függvényhez tartozó stackframe-en? Egy ilyen cím felülírásához milyen hosszú (bájtban mérve) input-ra lenne szüksége?

[+] Egy megfelelő felhasználói input segítségével a támadó túlcímzi a 64 bájtos buffert például úgy, hogy szomszédos memóriaterületeket ír felül. Legegyszerűbb esetben ez a cím az aktuális stack frame-hez tartozó visszatérési cím, ami EBP+4-en helyezkedik el. (4p)

[+] Mivel tudjuk, hogy 32 bites architektúráról van szó, ezért a 64 bájtos bufferíráson felül 4 bájt szükséges az előző stack frame bázis pointerének (Saved Frame Pointer) felülírásához, illetve 4 bájt kell, hogy a visszatérési címet is felülírjuk. Tehát összesen 64+4+4=72 bájtra van szükség. (3p)

Más jellegű támadások esetén (pl.: Return-to-Libc with EBP lifting) ennél több bájt szükséges, de indoklás nélkül vagy hibás indoklással a válasz nem fogadható el.

4.
A feladat egyértelműen leírja, hogy a támadó optimalizálja a támadását, ezért elég távolról közelíteni a dolgot. Minden egyes lekérdezés egy bit információt ad ki. Csak azt kell megvizsgálni, mennyi bit információra van szükség.

a.) Az ABC 26+26+10 =62 karakter (elrontotta kb. a dolgozatok 25%-a). Ez log_2(62)*8 bitet jelent és terészetesen ugyanennyi lekérdezést. Sokak számára log_2(62)=6, véleményünk szerint ez nem jó, mert log_2(64)=6. log_2(62) csak megközelítően 6. Kérjük ezt megfelelően jelölni, hisz a precizitás a mérnök erénye. Szőrszálhasogatás

lenne arról értekezni, hogy mivel ez törtszám, melyik mellette levő egész szám a helyes a feladat szempontjából, ezért ennél jobban nem vártuk el a megoldást.

b.) Az MD5 hash outputja 128 bit. 128 lekérdezés szükséges. (szövegesen is kiírta a választ – ahogy a középiskolában is elvárják ~20%)
Természetesen a feladat bonyolítható, de ekkor gyakoriak az elszámolások (pl. 8*16=108, 128 bit 8 biten = 8 karakter, stb.) Minta a bonyolításra: a 128 bit az 16 byte, adat, amit 32 hexa karakteren kódolunk. a 32 karakter minden egyes példánya 4 bit információt tartalmaz. 32x4=128. (meglepően visszajutottunk az eredeti felálláshoz, ám akik ezt a számítást megpróbálták megcsinálni, kb. 60% eséllyel a végére elhibázták és nem 128-at kaptak).

c.) A formátum bonyolult bemutatása és elemzése nem lényeges a feladat megoldásához, de jól teszteli a hallgató tudását és lényegrelátását, továbbá egy teljesen valós életbeli megoldásra nyújt rátekintést. A lényeges, hogy a brute force támadáshoz szükséges a hash is és a salt is. A salt 6*8 bit adattartalmú, a hash, mint írtuk 512 bites. 48+512=560. 560 lekérdezés szükséges.
Ennél a feladatnál volt néhány majdnem tökéletes megoldó, aki a hash rész adattartalmát úgy számolta, hogy 86 karakteren van Base64 kódolva, tehát 86*6 bit az adattartalom. Ez természetesen nem igaz, a feladatban jeleztük, hogy a hash függvény 512 bitet csinál, a 86*6=516 bit, a Base64 kódolás 4 bitet nem fog használni. A támadó természetesen tudja a formátumot, így a 4 bitet ő sem fogja lekérdezni feleslegesen. (a 4 bitnyi hely egyfajta padding itt)

Többek azt írták, hogy egy lekérdezés is elég, hisz lekérdezik és megvan. Ők nem olvasták el, hogy BLIND támadásról van szó, vagy nem hallottak róla.
Számos esetben az eredmény 2^570, 2^670, stb. lett. Nem tudjuk elképzelni, hogy valaki egy életszagú feladatnál hogy adhat meg olyan számot, aminek a nagyságrendjébe se gondol bele. Hasonlóképpen számos megoldás esetében 0-10 közötti számok jöttek ki. Elképzelésünk sincs, hogy mit gondolt a feladatmegoldó, milyen kérdéseket lehetne feltenni, hogy pár kérdésből kijöjjön egy több karakteres jelszó.
Többen voltak olyanok is, akik a támadás során minden karaktert, vagy hexa karaktert stb. „végigpörgettek", azaz minden lehetőséget kipróbáltak az adott elemre bináris keresés helyett. Ez természetesen nem jó, sokkal hosszabb lenne az optimális esetnél.
Megjegyzés: Az „optimális lekérdezés" elkészítése egyébként nem egyszerű. A b.) esetben pl. nincs specifikálva, hogy nagybetűs, vagy kisbetűs a hash hexadecimális reprezentációja, de elő lehet állítani megfelelő SQL parancsot pl. betu="A" or betu="a". A feladat megoldásához nem kértük az algoritmus leírását, az egyszerűség érdekében.
Rövid megoldás amit elfogadunk:

a.) a jelszó adattartalma log_2(64)*8 bit, felkerekítve egészre ennyi lekérdezés elég, hiszen minden lekérdezéssel az optimális támadó egy bit információhoz jut

b.) az előző feladat szerinti indoklással 128 bit információ van a hashben, ezért 128 lekérdezés kell

c.) a támadónak meg kell szerezni a hash-t és a salt-ot is. A Salt 8x6 bites, a hash 512 bites, 560 lekérdezés szükséges.

5.

1.

a. iptables –A FORWARD –p tcp –i eth1 –o eth2 –s 10.0.0.0/24 –d 10.0.1.1 –dport 80 –j ACCEPT

b. iptables –A FORWARD –p tcp –i eth2 –o eth1 –s 10.0.1.1 –sport 80 –d 10.0.0.0/24 –j ACCEPT

2.

a. iptables –A INPUT –p tcp –s 10.0.0.0/24 –dport 22 –J ACCEPT
b. iptables –A INPUT –p tcp –dport 22 –J DROP
c. iptables –A OUTPUT –d 10.0.0.0/24 –J ACCEPT
d. iptables –A OUTPUT –J DROP

3.

a. iptables –t NAT –A PREROUTING –p tcp –d 195.1.2.3 –dport 80 –to 10.0.1.1:80 –J DNAT
b. iptables –A PREROUTING –p tcp –d 195.1.2.3 –dport 443 –to 10.0.1.1:443 –J DNAT

4.

a. iptables –A FORWARD –dport 389 –j LOG

5.

a. Külső interfészről nem érkezhet csomag belső címes feladóval
b. Külső interfészről nem érkezhet csomag DMZ címes feladóval

6.

a.) u1 és u2, mert u1 és a g1 csoport tagjai írhatják az alkönyvtárat
b.) d1-be u1 és a g1 írhat, tehát csak u1 és u2 jön szóba, ők mindketten hozzáférnek a f4
fájlhoz, tehát u1 és u2.
c.) g1 lesz, mert csoport setgid jel van az alkönyvtáron
d.) Az összeset, mert a root felhasználó speciális jogú
e.) u1, ő a tulajdonosa (és a root)