6 Assignment 6 (11 Points)

Hinweis:	Abgabe in $\{2, 3, 4\}$-er Gruppen.
Abgabe:	**03.06.2017, 23.59**
Email:	Betreff "[Compsec] Ex6'
	(bitte nur .pdf oder .txt, kein .doc, .jpeg, etc)
	Source code: bitte inkl. `signify` Signatur

Exercise 17 (Honeywords **(3+2 Points)**).
Consider two users Alice and Bob, both having accounts at three web sites `1.example.org`, `2.example.org` and `3.example.org`. Alice chooses to use three separate passwords for 1, 2 and 3 while Bob wants to use the same.

1. Illustrate how websites 1 and 2 could implement *Honeywords* [1] to protect Alice and Bob from password cracking. In particular,
   - How should decoy passwords be constructed?
   - How many decoy passwords should be generated?

   Explain your decisions. (If you need to, you may make assumptions on how Alice's and Bob's passwords look like on 1 and 2 - for example enforced by the password registration process.) **Solution:** As we discussed in the tutorials there is no particular "correct" solution to this exercise. There are multiple solutions possible.

   Both websites (1 and 2) are using a take-a-tail approach when Alice and Bob first register or update their passwords, and a tail-tweaking approach otherwise. We do not know anyting about website 3 and make the conservative assumption that it does not use honeywords. We also offer a special option in the user interface to let Bob input a password at website 2 (1) that was previously generated at website 1 (2) using this approach.

   In the take-a-tail approach, each user is given a uniformly at random generated three-digit number in $\{0, \ldots, 9\}^3$, which is appended to the user-chosen password. Users are not allowed to choose these digits by themselves or select among a list of three-digit numbers. This assures us that each tail is truly uniformly distributed, which we need in calculating an adversaries success probability in the second part below. In the tail-tweaking approach we expect users to input a password that was previously generated using the take-a-tail approach. Honeywords are then created (in both cases) by randomizing the last three digits of the password.

   For the number of honeywords (including users true passwords) we select 100, which is between the numbers 30 and 300 that were suggested in the original paper.

2. Using your construction, analyze the success probability of an adversary impersonating Alice and Bob to 3, who successfully breaches (excluding the Honeychecker)
   - only `1.example.org` or `2.example.org`
   - both.

   Note: you must assume that the process *how* decoys are generated is publicly known (no security through obscurity). **Solution:**

   Alice: If all three passwords are unrelated then it would not improve an adversaries guessing strategy at all. For concreteness however, let's assume that the three passwords are related and of the form pwx_1 on website 1, pwx_2 on website 2 and pw on website 3 where x_1 and x_2 are the randomly generated three-digit numbers explained above. In this case, a breach of website 1 or 2 will give an adversary enough information (remember that the process how honeywords are generated is assumed to be public information) to find the base of the password (pw) and try that on website three. The success probability will thus be 1 in one try.

   Bob: Bobs password is of the form pw$x$ where x is a randomly generated three-digit number, and this password is used on all three sites.

   If one website, say 1, is breached then the adversary is left with 100 candidate passwords that are all equally likely. The probability to select the right password in i tries is thus $i/100$ on website 3.

   The adversary could additionally use website 2 to test if a guess is correct or not before attempting to log into website 3. However we have not made any assumptions how website 1 or 2 treat incorrect entries (blocking the user or setting of an alarm).

   If both websites are breached then the adversary can build the intersection of both sets of honeywords. Let S_1 be the set of honeywords of website 1 and S_2 the set of honeywords of website 2. The correct password must be in $S_1 \cap S_2$.

   We now want to calculate the number of decoys $i \neq x$ in $S_1 \cap S_2$. In order to calculate this number we define for each $i \in \{000, 001, \dots, 999\} \setminus x$ the random variable

$$X_i := \begin{cases} 1 & \text{, if pw}i \in S_1 \cap S_2, \\ 0 & \text{, else.} \end{cases}$$

For any X_i the probability that this possible decoy is included in both sets is then

$$\Pr[X_i = 1] = \left(\frac{99}{999}\right)^2,$$

since are at most 999 decoys possible, we did select 99 of them and this particular decoy had to be selected for S_1 as well as S_2. The total number of decoys is then $X_{000} + \ldots X_{999}$, and the expected value is

$$E[\sum_i X_i] = \sum_i E[X_i] = 999 \cdot \left(\frac{99}{999}\right)^2 = \frac{99^2}{999} \simeq 9.81.$$

So the adversary is left with about $9.81 + 1$ passwords to guess from, which gives a guessing probability of 0.09.

Note that if the number of digits appended to the password increases this probability quickly approaches 1. On the other hand, if the number of digits is too small then there won't be enough digits to get the desired number of distinct honeywords (and typos will more often lead to false positives). The optimum number of digits will give precisely the number of desired honeywords.

Exercise 18 (x86 assembly recap **(2 Points)**).
Write a simple, minimal hello-world program in x86 assembly. Write it yourself and do not only disassemble a C-program. Your program should compile with `gcc` on your OpenBSD virtual machine (please provide a Makefile) and should run without any errors.

Solution: We provide two solutions: one linking to the standard C library and one standalone. When linking with `libc` we need to define the main symbol that libc will pass control to.

```
1  hello:   .ascii "Hello world\n"
2  .text
3  .global main
4  main:
5     pushl       $12             # strlen(hello)
6     pushl       $hello
7     pushl       $1              # stdio
8     pushl       $1              # for cdecl compliant syscalls
9     mov         $4, %eax        # syscall number 4 (write)
10    int         $0x80
11
12    mov         $1, %eax        # syscall number 1 (exit)
13    pushl       $0              # return value for exit
14    pushl       $1              # for cdecl compliant syscalls
15    int         $0x80
```

If we do not want to link to libc (because we don't need it) we need to define the _start symbol instead. We additionally have to include a section (.note.openbsd.ident) that indicates to the kernel that this program is a native OpenBSD program.

```
1  .section ".note.openbsd.ident", "a"
2                  .p2align 2
3                  .long    8
4                  .long    4
5                  .long    1
6                  .ascii   "OpenBSD"
7                  .long    0
8                  .p2align 2
9
10 .section .data
11 hello: .ascii "hello␣world\n"
12 .section        .text
13 .global         _start
14 _start:
15   pushl         $12              # strlen(hello)
16   pushl         $hello
17   pushl         $1               # stdio
18   pushl         $1               # for cdecl compliant syscalls
19   mov           $4, %eax         # syscall number 4 (write)
20   int           $0x80
21
22   mov           $1, %eax         # syscall number 1 (exit)
23   pushl         $0               # return value for exit
24   pushl         $1               # for cdecl compliant syscalls
25   int           $0x80
```

Exercise 19 (Case study μ-`shout` (II): Secure C Coding (**2 Points + 1 Bonus**))**.** We want to improve our μ-`shout` prototype. Fix all of your previous programming mistakes (if any). Additionally, have a look at the *SEI CERT C Coding Standard* at

`https://www.securecoding.cert.org/`

and adjust your implementation by following at least one rule or recommendation. For your submission: document wich programming mistakes you fixed and all the changes you made according to the secure C coding rules (recommendations), as well as the relevant rules (recommendations).

In case you followed any tutorials in order to program your previous exercises please have a look at [2] as well.

Exercise 20 (Keeping your systems secure (**Bonus: 1 Points**))**.**
Are there any new vulnerabilities for your Debian or OpenBSD system since last week (27.05.2016 at 23.59)? If so: state one, **name the programming mistake**,

decide if you are affected or not, and report if there are any known work-arounds or patches.

References

[1] Ari Juels and Ronald L. Rivest. Honeywords: making password-cracking detectable. In Proceedings of CCS '13.

[2] Tommi Unruh, Bhargava Shastry, Malte Skoruppa, Federico Maggi, Konrad Rieck, Jean-Pierre Seifert, Fabian Yamaguchi. *Leveraging Flawed Tutorials for Seeding Large-Scale Web Vulnerability Discovery*. arXiv:1704.02786 [cs.CR].