Computer and Network Security
8th of August 2015

- This exam consists of 7 questions with subquestions. Unless indicated otherwise, every subquestion counts for 10 points.
- Mark every page with name and student number.
- Use of books, calculator, or additional course material is prohibited.
- Always explain your answers. At the same time, keep your answers short and to the point. Do not use pencil or red ink.

---

1. **True/False with Reason (10 points total).** For each of the statements below, state whether they are true or false. Explain your answer with a short justification, at most 2 sentences long.

   1. Non-repudiation: Alice sends messages to the bank with the following fields:
      - Alice's account number;
      - beneficiary;
      - amount to be transferred;
      - free-form description;
      - signature of preceding fields with Alice's private key.

      each such message is encrypted with bank's public key

      The bank claims that Alice has transferred for a total of 1236 euros. Alice denies and says she only transferred 618 euros. Alice's denial is plausible.
   2. A misuse-based intrusion detection system suffers less from false positives than an anomaly-based system.
   3. Alice regularly sends messages to Bob using an encryption scheme that is somewhat similar to WEP as discussed in the lectures, except that initialization vector is 64 bits. She picks the initialization vector sequentially, that is the first message has IV=0, the second IV=1 and so on. →This is just as secure as using a cryptographic-strength random function to determine IVs.
   4. Based on the round-trip time, a carefully chosen timeout, and a threshold on the number of concurrent connections, SYN Cookies computes a TCP initial sequence number that defeats TCP SYN flooding attacks.
   5. ARP poisoning requires an attacker to have access to the victim's network segment.

2. **Passwords**

   A popular password policy is to require capital letters in passwords. Research has shown that when people are asked to use capital letters in passwords, they capitalise exactly 1 letter in their passwords. Therefore you can see this policy as a requirement to type one extra key (SHIFT) in your password.

   (a) Let us assume first that passwords are required to be exactly 8 characters long and only lowercase [a-z] is allowed. How many possible passwords are there?

   (b) Now we introduce the policy. We require the same number of key presses, but exactly one of them should be for the SHIFT key (so the new policy is a sequence of a-z characters of length, except that one of them is a capital). (i) How many possible passwords do we have now? (ii) Which policy is more secure?

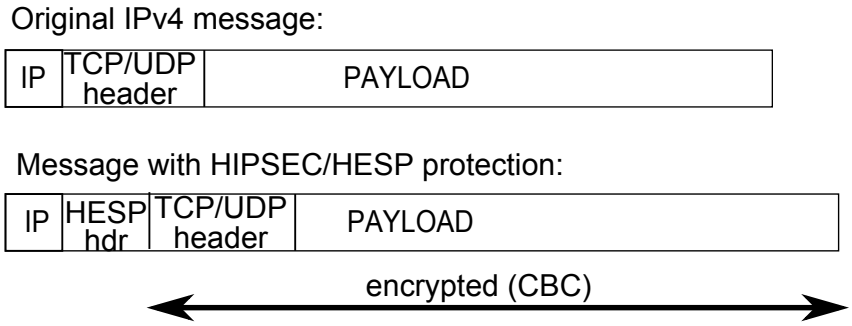

Figure 1: HIPSEC protection: all of the transport layer (TCP/UDP) header and contect are encrypted

3. **crypto, networks, network protection**

   (a) An IDS has a precision of 0.999. A network administrator complains that (s)he receives 10 false positives per week. How many true positives does the IDS generate?

   Alice and Bob communicate over TCP via **HIPSec** ("Herbert's IPSec-like protocol"), with **HESP** (Herbert's Encapsulating Security Payload), using an encryption scheme with **CBC** (code block chaining). There is no separate Authentication Header (AH).

   (b) HIPSec and HESP were inspired by IPSEC and ESP. **(i)** Explain how (real) IPSec with ESP works (sketch what happens when packets are transmitted *and* when they are received, and show roughly what the packets look like), **(ii)** Explain CBC.

   HIPSec with HESP work as follows. Prior to communication, the communicating machines establish a security association that defines security parameters such as the key. Now all of the IP datagram's content will be encrypted (using CBC) as illustrated in Figure 1.

   Assume that an attacker, Carl, manages to get an account on the machines used by Alice and Bob. We will call the presence on Alice's machine Carl_A, and on Bob's machine Carl_B. Carl can sniff all traffic on the network of either machine (his own machine is on Alice's unprotected Wifi network), but he cannot interfere with Alice's transmission, he cannot see Alice's memory or her packets prior to encryption, and does not know the key agreed upon in the security association. Carl can send UDP and TCP packets from Alice's machine and raw IP packets (for instance, from his own machine).

   (c) Will Carl be able to read Alice's data? If so, will he be able to read *all* of the data? If not, why not? Explain in detail!

4. **Memory errors** A SUID root program has the following code:

```
1  #include<stdio.h>
2  #include<string.h>
3
4  int main(int argc, char **argv) {
5     char *p, result[512];
6     char hostname[128], password[128], username[128];
7
8     if (argc != 4) {
9       fprintf(stderr, "bad arguments\n");
10      return -1;
11    }
12
13    strcpy(username, argv[1]);
14    strcpy(password, argv[2]);
15    strcpy(hostname, argv[3]);
16
17    p = result;
18    strcpy(p, "ftp://"); p += 6;
19    strcpy(p, username); p += strlen(username);
20    strcpy(p, ":");      p += 1;
21    strcpy(p, password); p += strlen(password);
22    strcpy(p, "@");      p += 1;
23    strcpy(p, hostname);
```

```
24
25      printf("%s\n", result);
26      return 0;
27  }
```

(a) Explain why this code is vulnerable and how you can exploit it to launch a shell. Assume that all possible protections (such as address space layout randomisation, bound checking, canaries and non-executable stack) are disabled and that variables are pushed onto the stack in the order in which they are declared with no additional padding. You should describe your inputs in enough detail to make it work (and not crash the program) but there is no need to give the exploit code or shellcode.

(b) After finding his entire user database posted on Pastebin by the famous CLF group, the system administrator finds the vulnerability and attempts to fix the program by replacing the first three string copies (lines 13-15) with "safe" versions[1]:

```
1       strncpy(username, argv[1], 128);
2       strncpy(password, argv[2], 128);
3       strncpy(hostname, argv[3], 128);
```

Can you still exploit the program now? If so, how? If not, why not?

(c) Assuming the stack is made non-executable, is it still possible to exploit the program? If so, what do you need to change?

(d) Assuming (random-value) stack canaries are also used, is it still possible to exploit the program? If so, what do you need to change?

5. **Shellcode**

Suppose you have the following piece of shellcode:

```
1   address        opcodes                    assembly
2   -------        ---------                  --------------------------------
3   80483f8:        eb 1f                     jmp    8048419 <shellcode+0x21> ; <-- relative jmp
4   80483fa:        5e                        pop    %esi
5   80483fb:        31 c0                     xor    %eax,%eax
6   80483fd:        88 46 07                  mov    %al,0x7(%esi)
7   8048400:        89 76 08                  mov    %esi,0x8(%esi)
8   8048403:        89 46 0c                  mov    %eax,0xc(%esi)
9   8048406:        b0 0b                     mov    $0xb,%al
10  8048408:        89 f3                     mov    %esi,%ebx
11  804840a:        8d 4e 08                  lea    0x8(%esi),%ecx
12  804840d:        8d 56 0c                  lea    0xc(%esi),%edx
13  8048410:        cd 80                     int    $0x80
14  8048412:        31 db                     xor    %ebx,%ebx
15  8048414:        89 d8                     mov    %ebx,%eax
16  8048416:        40                        inc    %eax
17  8048417:        cd 80                     int    $0x80
18  8048419:        e8 dc ff ff ff            call   80483fa <shellcode+0x2>
19  804841e:        2f 62 69 6e 2f 73 68 00 .string "/bin/sh"
```

(a) You want to use this piece of shellcode remotely, but you have no convient way to control stdin. However, all you need to do to completely pwn the system is to execute `/usr/local/bin/l33t`. Modify the shellcode in such a way that it executes `/usr/local/bin/l33t` rather than `/bin/sh`. Include in your answer a list of changes you made.

(b) You have tried to use your shellcode to overflow a buffer used to store the 'name' field of a web form. However, you failed to pwn the system. Instead you receive an error message:

```
1   <html>
2   <head><title>ERROR!</title></head>
3   <body>
4   <h1>ERROR!</h1>
5   <p>
6   Numeric digits ('0'-'9') are not allowed in the name field!
7   Please fix it and try again.
8   </p>
9   </body>
10  </html>
```

Modify the shellcode so as not to contain any numeric digits (ASCII codes 0x30-0x39). A quick look in the "Intel 64 and IA-32 Architectures Software Developer's Manual Volume 2B" reveals that no opcodes other than `XOR` use these bytes[2].

[1] *strncpy*(src, dst, n) copies up to n characters. If *dst* is longer than n, it will not end *src* with a terminating null.
[2] Actually, this is a minor simplification to keep things clear. In reality, there are a few more that use these bytes.

6. **Networks**

   (a) What does the following pcap dump represent (complete your answer as accurately as possible)?

```
21:03:59.711106 snap 0:0:0:8:0 brutus.net.53 > host201.dopey.org.21: F 0:0(0) win 2048 (ttl 48, id 55097)
21:04:05.738307 snap 0:0:0:8:0 brutus.net.53 > host201.dopey.org.21: F 0:0(0) win 2048 (ttl 48, id 50715)
21:05:10.399065 snap 0:0:0:8:0 brutus.net.53 > host202.dopey.org.21: F 0:0(0) win 3072 (ttl 49, id 32642)
21:05:16.429001 snap 0:0:0:8:0 brutus.net.53 > host202.dopey.org.21: F 0:0(0) win 3072 (ttl 49, id 31501)
21:09:12.202997 snap 0:0:0:8:0 brutus.net.53 > host24.dopey.org.21: F 0:0(0) win 2048 (ttl 52, id 47689)
21:09:18.215642 snap 0:0:0:8:0 brutus.net.53 > host24.dopey.org.21: F 0:0(0) win 2048 (ttl 52, id 26723)
21:10:22.664153 snap 0:0:0:8:0 brutus.net.53 > host3.dopey.org.21: F 0:0(0) win 3072 (ttl 53, id 24838)
21:10:28.691982 snap 0:0:0:8:0 brutus.net.53 > host3.dopey.org.21: F 0:0(0) win 3072 (ttl 53, id 25257)
21:11:10.213615 snap 0:0:0:8:0 brutus.net.53 > host102.dopey.org.21: F 0:0(0) win 4096 (ttl 58, id 61907)
21:11:10.227485 snap 0:0:0:8:0 host102.dopey.org.21 > brutus.net.53: R 0:0(0) ack 4294947297 win 0 (ttl 25, id 38400)
```

   (b) CLF president Dmitri wants to launch a denial of service attack on Herbert's website `www.hjb.nl`. The website runs on a state-of-the-art operating system. To launch the attack, Dmitri has (only) a dozen hosts (bots) spread across the world at his service. He has several methods to pick from:

   1. Dmitri sends a lot of ICMP echo requests to `www.hjb.nl` from the bots directly – with (random) spoofed IP addresses;
   2. Dmitri sends a lot of ICMP echo replies to `www.hjb.nl` from the bots directly – with spoofed IP addresses (chosen from a collection of legitimate addresses, collected earlier);
   3. Dmitri sends a lot of ICMP echo requests to many other hosts with the IP address of `www.hjb.nl` as (spoofed) source address.

   The objective is to make it very difficult to stop the attack even if the ISPs cooperate with Herbert. Which method will Dmitri use and why?

7. **Web attacks**

   A website contains the following code:

```
<h3>Please let me know what you think about my site:</h3>
<form action="simple.py" method="post">
<textarea name="comment" rows="10" cols="50">
(Type text here)
</textarea>
<input type="submit" value="Test your input">
</form>
```

   The script `simple.py` stores the script in a file `comments.txt`, as follows:

```
# get the comment parameter
if form.has_key("comment"):
        comment = form["comment"].value;
else:
        comment = ""

# and append it to the comments.txt file
f = open("comments.txt", "a")
f.write("""<p>User wrote:</p><pre>%s</pre>""" % (comment))

# tell the user the comment was saved and where to see all comments
print ("""<p>Your comment has been saved!</p>
       <p>Go to <a href=\"view.py\">the forum</a> to view all comments</p>""")
```

   Assuming the HTTP request contains a cookie that holds the username and password, the script `view.py` reads the file `comments.txt` and displays them as follows:

```
f = open("comments.txt", "r")
comments = f.read()

print ("<p>These are the comments so far:</p>")
print "<p>" + comments + "<p>"
....
```

   (a) The site is vulnerable. Explain the vulnerability.

   (b) Will the vulnerability go away if the output of the `view.py` script goes to a separate frame in the user's browser? If so, why? If not, why not?