Computer and Network Security

24th of March 2014

- 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.
- Do not panic!

---

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. On systems where the `syscall + ret` gadget is in a fixed, known location, we can use sigreturn oriented programming (SROP) for exploitation if (a) we can control the stack and the contents of the RAX register, (b) know *any* stack address, and (c) are able to divert the control flow of the program to the `syscall + ret` snippet. Specifically, we do not need *any* other gadget.
   3. A misuse-based intrusion detection system suffers less from false positives than an anomaly-based system.
   4. Alice regularly sends messages to Bob using an encryption scheme that is somewhat similar to WEP as discussed in the lectures, except that the initialization vector is 128 bits and the initialization vector has to be picked sequentially by each host, that is the first message sent by a host has IV=0, the second IV=1 and so on. A recipient will check whether the IV is strictly increasing. →This scheme is more secure than WEP.
   5. An idle scan is a good way to "probe behind a firewall". Using a (mostly idle) machine as intermediate, it allows us to scan machines that we may not be able to reach otherwise.
   6. 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.
   7. ARP poisoning requires an attacker to have access to the victim's network segment.
   8. Cross-site scripting attacks (XSS) require users to click on attacker-provided links. The link could be sent by the attacker to the victim (e.g., in a spam message), or it could be stored at a (possible benign) website. Either way, you are safe as long as you do not click it.

9. The x86 architecture is great for ROP attacks, because of the variable-size instructions which allow an attacker to land into “the middle” of an instruction. In this way, attackers may find even more gadgets. However, this trick no longer works if CFI is used.
10. Cybercriminals use fast flux to avoid exposing the real malicious servers behind a (potentially large) number of zombies that only serve to forward the requests and shield the server.

2. **Crypto**

   (a) You would never know it by looking at him, but the student sitting next to you (his name is Bob, by the way), just invented a cryptographically secure pseudorandom bit generator that takes a seed of 128 bits. Show how he can turn this into a symmetric cryptosystem with a key of 128 bits.

   (b) Assume that in the encryption scheme, Bob’s keystream is never re-used. Can we still fruitfully use the WEP-cracking trick where we reconstructed a few bytes of the keystream – because we *know* certain patterns exist in the traffic (like ARP requests, etc.) – and use it to decrypt other traffic?

   (c) Public key certificates are convenient, but as we saw in class, revocation of certificates may be problematic. A typical solution is to publish revocation lists (lists of revoked certificates), but they are cumbersome and often not immediate. Ever the clever one, Bob suggests to handle revocation by putting an expiration date in a public key certicate. When the expiration date is reached, the certicate is no longer valid and will not be renewed. What are two disadvantages of this approach?

3. **Memory errors**

   Trying to make some money on the side, Herbert is selling software to the police to keep track of traffic violators. The core of the (SUID) program consists of code to add or delete license plate numbers to an in-memory list, as shown in Fig. 3. Any police officer can connect to the server, and type either ‘i’ or ’d’ (to insert or delete, respectively) followed by the 8-character long license plate.

   (a) As you can imagine, Herbert makes oodles of money with this advanced code. However, although the software is protected with canaries, it does not take long for the system to get pwned by rogue cops who use it to spawn a shell and get root access to classified data. Herbert denies any problems with the software, and the police asks *you* to have a look at it. You immediately find a gaping vulnerability. What is it and how would you exploit it? (Assume for now that the stack is not randomized and there is no DEP.)

   Grudgingly, Herbert fixes the problem by making every buffer in the original code MAXSTR in size. He also addresses the complaint that instead of ’i’ some users often accidentally type ’insert’, or ’ins’. The program now accepts any word that starts with ’i’ and any word that starts with ’d’. The new code for `main` is shown below.

```
int main (int argc, char *argv[]) {
  char *logMsg, action[8], license[MAXSTR];
  int n, fd = ... /* some socket */
  struct node *list;
  for (;;) {
    logMsg = newlogMessage(/*...*/);
    read (fd, &action, MAXSTR-1);      // read 'i[nsert] or d[elete]
    n=read (fd, license, MAXSTR-1);    // read license plate
    license[n]=0;                      // ensure null termination
    printf (logMsg);
    if (strncmp(action, "i", 1) == 0) list = insert (fd, list, license);
    else if (strncmp(action, "d", 1) == 0) list = delete(fd, list, license);
    else exit (0);
  }
  return 0;
}
```

   (b) Can this code still be exploited? Describe how!

   (c) The police are fed up and decide to turn on ASLR and DEP. Will it help?

```c
char *newlogMessage() { ... }
#define MAXSTR 64

struct node {
  char license[8];
  struct node *prev;
  struct node *next;
};

struct node *insert (int fd, struct node *list, char *license) {
  struct node *node = (struct node*) malloc (sizeof(struct node));
  node->prev = NULL;
  node->next = list;                       // insert at head of list
  if (list != NULL) list->prev = node; // add backpointer
  strcpy (node->license, license);      // fill string with license plate
  printf ("Added␣license␣plate␣%s␣at␣node␣%p\n", license, node);
  return node;
}

struct node *delete (int fd, struct node *list, char *license) {
  struct node *lp=list;
  while (lp) {
    if (strncmp(license, lp->license, 8) == 0) {           // found the license plate
      if (lp->prev != NULL) lp->prev->next = lp->next;
      else list = list->next;
      if (lp->next != NULL) lp->next->prev = lp->prev;
      free (lp); break;
    } else lp=lp->next;
  }
  return list;
}

int main (int argc, char *argv[]) {
  char action, *logMsg, license[MAXSTR];
  int n, fd = fileno(stdin);
  struct node *list;

  for (;;) {
    logMsg = newlogMessage(/*...*/);
    read (fd, &action, 1);
    n=read (fd, license, MAXSTR-1);
    license[n]=0; // make sure license plate string is null terminated
    printf (logMsg);
    if (action == 'i') list = insert (fd, list, license);
    else if (action == 'd') list = delete(fd, list, license);
    else exit (0);
    //dumplist(list);
 }
  return 0;
}
```

Figure 1: The core of Herbert's advanced anti traffic violation software

4. **Shellcode**

   Consider the following x86 assembly code fragment that is intended to work on a server that reads in strings from the network:

```
jmp ahead
back:
  popl %esi
  xorl %eax, %eax

  # ... missing instruction ...

  leal (%esi), %ebx    # place address of string '/bin/sh' in %ebx
  movl %ebx,8(%esi)    # ... and copy it to this memory location (for argument two)
```

```
10    movl %eax,12(%esi)   # ... and store null into this location (for argument three)
11
12    movb $0x0b,%al       # syscall 0x0b represents execve
13    movl %esi,%ebx       # again: argument one -> /bin/sh
14    leal 8(%esi),%ecx    #        argument two -> pointer to /bin/sh
15    leal 12(%esi),%edx   #        argument three -> pointer to NULL
16    int  $0x80
17
18  ahead:
19    call back
20
21    .ascii "/bin/sh#AAAABBBB"
```

(a) Clearly, this is shellcode that tries to execute `/bin/sh` by means of the `execve` system call. The string `/bin/sh` will be the argument to `execve`. Explain how the shellcode finds this address.

(b) Why do you think the shellcode writer put "AAAABBBB" after the `/bin/sh` string (after all these bytes will all be overwritten - could the attacker could have omitted them)?

(c) The shellcode will not work. Why not? How would you fix it by adding a single instruction on line 6?

5. **Botnet Infiltration**

You are analyzing an unstructured UDP-based Peer-to-Peer botnet which has recently appeared in the wild. Your goal is to infiltrate and disable the botnet.

(a) **Peer list poisoning**
Your first thought is to disrupt the botnet by poisoning the peer lists of all the bots. The code used by bots to add a peer list entry is shown below. Uninitialized peer list entries are zeroed out, and the first such entry marks the end of the list. The parameters for the `add_to_list` function are copied from a peer announcement message, which you can push to a bot at will.

IP (4 bytes)	Port (4 bytes)

Figure 2: A peer announcement message.

```
typedef struct {
  unsigned int ip;
  unsigned short port;
} pl_entry;

pl_entry peerlist[512];

int add_to_list(unsigned int ip, unsigned int port)
{
  int i;
  for(i = 0; i < 512; i++) {
    if(peerlist[i].ip == 0) {
      break;
    } else if(peerlist[i].ip == ip && peerlist[i].port == port) {
      return -1;
    }
  }

  if(i == 512) {
    return -1;
  } else {
    peerlist[i].ip = ip;
    peerlist[i].port = port;
  }

  return 0;
}
```

Figure 3: The procedure for adding a new peer list entry.

To keep the logistics simple, you would like to use a single IP/port pair to launch your poisoning attack. Is there any way to poison all peer list entries in the network using just one IP and port? If so, how, and if not, why not?

(b) **Remote cleanup**
The botmaster commands the bots by pushing RSA-signed messages into the botnet, which the bots propagate using gossiping. Whenever a bot receives a command from one of its peers, it forwards it to all its other peers.

IP (4 bytes)	Port (4 bytes)	Signature (256 bytes)

Figure 4: A command announcement message.

A command announcement contains the IP and port of a bot which can be contacted to fetch an executable payload. Each bot will fetch this payload and run it to complete the command. The bot to host a payload is selected by the botmasters and, unfortunately, tthe botmasters never voluntarily select you to serve a payload. You may be able (using the police and a warrant) to obtain the IP address of these payload-hosting bots for yourself, but that process takes weeks, and by that time all the bots have already downloaded the payload. Even so, is it possible to abuse the command announcement mechanism to propagate a cleanup program to all bots? Explain your answer.

6. **Networks.**
   (a) One difference between IPv6 and IPv4 is that IPv6 no longer supports fragmentation. If a datagram is too large for a link, it will simply be dropped and an ICMP error message returned to the source. Explain in what way this is good for security (7 points). Also explain in what way it could be abused by attackers (3 points). Use no more than 3 sentences for each answer.
   (b) Explain briefly the idea of "desynchronization" in the Joncheray hijacking attack.

7. **Web attacks**
   (a) Can an attacker use cross site request forgery (CSRF) to obtain a user's cookies corresponding to a bank? Explain.
   (b) Fed up with all CSRF attacks on its clients, a bank decides to change all its scripts to use only the POST method instead of GET. In what way will this make life more difficult for attackers? Can they still perform the CSRF attack?