Exercise Sheet 5

Introduction to IT-Security

Submit your solutions via Gitlab.
Deadline: *Wednesday, January 14th, 09:00 a.m. CET 2026*

Basic Network Security

1. Look at the file `src/traffic.dump`.

   The file contains network traffic in `pcap` format. Make yourself familiar with tools for analysis of this data (e.g. wireshark) and answer the following questions. Explain in detail how you were able to determine the values:

   (a) (1 point) Which application-layer protocols are contained in the traffic?

   (b) (1 point) What is the password of `tristank`?

   (c) (1 point) Who is the recipient of joyh's's email (`joyh@...`)?

   (d) (1 point) Who downloads the file `committee_test.c`?

2. (2 points) The Smurf attack uses `ICMP` ping messages for flooding a victim with network data. The Echo protocol (RFC 862) implements functionality similar to `ICMP` ping. The protocol supports `TCP` as well as `UDP` for transport.

   Is it possible to adapt the Smurf attack to run over (a) `UDP` or (b) `TCP`? Explain your results for both cases.

3. (8 points) Many attacks against the transport layer make use of modified TCP flags. Develop a program that can send TCP packets with modified flags to a given IP address and port *given as program arguments* by programming raw sockets. This means you are asked to compose the structure (header, checksum, etc.) yourself and send the packet using raw sockets.
   Important: You must invoke your program with root privileges to utilize raw sockets.

   Implement support for the following packet types and provide a log showing it use:

- *SYN packet*: `TCP` packet with only the `SYN` flag set
- *XMAS packet*: `TCP` packet with the `FIN`, `URG` and `PSH` flags set
- *FIN packet*: `TCP` packet with only the `FIN` flag set
- *NULL packet*: `TCP` packet with no flags set

```
send_packet.py [-h] [--syn | --xmas | --fin | --null] IP/DOMAIN PORT
```

4. (12 points) **Master:** A *Port Knocking Daemon* is sometimes deployed on firewalls to allow external access to a port upon receipt of a certain sequence of network packets. In this task you are asked to implement a challenge-response port knocker.

   1. Develop a server that monitors all UDP and TCP communication and has the following synopsis:
      `server.py [-h] --port INT --shared-key INT --num-knocks INT`

      Upon receipt of a UDP packet on specified port (`--port`), the server replies with another UDP packet that contains a numeric challenge c as raw data. From this time on the server records a specified number (`--num-knocks`) of TCP SYN packets.

      Once received, the server verifies if the recorded TCP packets have been sent to the correct sequence of ports. If so, print out the text "unlocked" and remain silent otherwise. Usually the server would open the requested port and allow the commmunication over that port.

   2. Write a program which transmits a sequence of TCP packets with only the SYN flag set to a number of ports (`--num-knocks`), which are determined as follows:

      $$p_i = 1024 + (\texttt{sha256}(k * c + i) \bmod 28657) \text{ for } i \in \mathbb{N}^+$$

      k is a shared key and c the challenge—both are numeric. The shared key can be provided as program argument, the challenge is received from the server in response to a UDP packet. The synopsis looks as follows:

      `client.py [-h] --shared-key INT --num-knocks INT IP/DOMAIN PORT`

   3. Test your implementations and show how to use both implementation in combination.

   Use of Libraries. In contrast to the previous task for this one you are allowed and explicitly adviced to use *scapy*[1].

[1] http://www.secdev.org/projects/scapy/