Lecture Network Security
Summer Term 2015

Jens Tölle, Wolfgang Moll
Jonathan Chapman, Laura Guevara, Martin Lambertz, Rafael Uetz

Assignment Sheet 2

Publication date:	Thu, 07.05.2015
Submission deadline:	Tue, 19.05.2015
Discussion of results:	Thu, 21.05.2015

Task 2.1 (theoretical): Reconnaissance in the SecLab

There are a bunch of computers in the *Network Security Laboratory (SecLab)*. When hackers try to break into a network, they start with reconnaissance of their target and try to collect as much information as they can.

Log in to the SecLab with your personal account that you should have received per email. Find out as much as you can about the lab: IP addresses, running services, operating systems, etc. What else can you find out? Can you find the *candy* (in form of recognizable strings) that we hid in the SecLab for you? Please, do not use tools or methods that may disturb or kill services in the lab.

Submit a list of tools you used, source code of self-made programs/scripts, and all information you gathered.

Feel free to discuss tools and methods with your fellow students on the mailing list (however, do not post your findings).

Task 2.2 (practical): DNS Sniffing

You have heard about weaknesses in the Domain Name System (DNS) in the lecture. You will be developing a DNS spoofer during this and the next assignment sheet. To start with, this task covers the sniffer part, which you will extend in the next assignment sheet to actually send fake responses.

Write a program that monitors all DNS-related UDP traffic and prints out the relevant protocol fields of requests and responses you see.

Submit the source code of your tool and a sample output for both requests and responses *recorded in the SecLab*!

Tips:

- You need super-user (root) privileges to monitor the traffic from other machines. Use `sudo -i` to become root.
- You can use any common programming language.
- If libraries/tools are missing, you are allowed to install them on your own. Just send us a short mail about what you installed.
- The *pcap* library is a good means for capturing network traffic.
- There are libraries, like e.g., *libnet*, that ease up parsing captured traffic.
- Structs and similar data types are useful for parsing, too.
- You don't have to write a tool that can deal with all possible DNS requests (like e.g., AAAA, MX). It is enough to handle type "A" requests as you see in the lab. Also, focus on UDP. We will ignore TCP-based DNS for this task.
- You can use tcpdump or wireshark/ethereal to verify your results.

Task 2.3 (practical): Hash Collisions

Let's build our own super-collider!
Write a small program that generates two independent random byte sequences with length at least 64 bytes and calculates the SHA256 digest on these sequences. Repeat this step until you found a collision on on the first four bits, i. e. the 4-bit prefix, of the hash values.
Now run this program five or more times and note how many tries you needed until a collision occurred. Repeat this step comparing the first 8, 12, 16 and 20 bits and create a plot indicating how many counts you needed until a collision occurred for each of these bit lengths. Add the average count needed for creating a collision on each of the prefixes and connect the consecutive averages with a line.
Your hand-in must include:

- The documented source code of the collider
- The data file(s) generated by the collider
- The documented plotting script
- The output of the plotting script as PDF or PNG file

Task 2.4 (theoretical): Designing Asymmetric Encryption Schemes

Part (a):
Bob has the idea to use the following asymmetric method to secure the transport of objects: To be able to securely transmit objects (e.g., chocolate or money), he uses a lockable box. Since it is not possible to exchange keys with the partner, Bob locks the box with a padlock. Only Bob has a key to this padlock. The locked box is sent to Alice by a parcel service. The lock provides integrity and privacy, i.e., nobody is able to see or change the content of the box. Alice cannot open the box, since she does not own a key to the padlock. Thus, she adds a second padlock to the box. Alice has the only key to this second padlock. The parcel service returns the box to Bob. This time the box is secured with two padlocks. Bob removes the first padlock (the one he has a key for) and sends the box back to his partner. Alice receives the box, removes the second padlock and opens the box.

Is this method secure? Does it also work with cryptographic means? Which problems could arise?

Part (b):
Bob believes in his method. Thus, he wants to send digital data using the same approach. Instead of padlocks and keys, Bob chooses a secret key and XORs every character of the clear text with one character of the key (key length is at minimum the length of the clear text). The partner also chooses his own, different secret key. The method is now performed as in Part (a). Does it work? Can confidentiality and integrity be assured? Would choosing different random keys for each message have an impact?

Task 2.5 (practical): HMAC

In the lecture you heard about HMAC. Now it is your task to create your own HMAC. Take the hash function of your choice and implement an HMAC using it. Use an appropriate block size for that hash function. To generate the key to be used, execute `name2key` on hellgate. Take the ASCII representation of the output padded to the block size with leading zeros as a key for your HMAC. Use the byte value `0x93` to construct an IPAD, and `0xA5` for the OPAD. Use your HMAC to compute a hash that authenticates this PDF file. Which hash function did you choose? Reason your choice.

Send via email:

- The source code of your HMAC
- The HMAC for this PDF document