Exam Advanced Network Security

Jaap-Henk Hoepman, Joeri de Ruiter

July 2, 2018

NOTE: READ THIS CAREFULLY: This exam consists of two alternatives. The first alternative is the regular exam for students that followed the course this academic year (with lectures by Joeri de Ruiter and Jaap-Henk Hoepman). The second alternative is *only* intended as the final resit exam for those students that followed the course the previous academic years (with lectures only by Jaap-Henk Hoepman). **Select the alternative that applies to you and only answer the questions in that alternative**.

Please answer your questions using the space provided on the exam sheet. Write legibly, and use proper sentences; an unreadable answer is a wrong answer. . . Answer questions concisely, but with sufficient detail and precision. Always explain your answers. Please leave space in the margin for correction marks. Use a separate piece of scratch paper for draft answers, private computations or remarks. **Write your name and student number on every page.**

You are not allowed to use books, notes, tablets, PCs and (smart)phones during the exam.

The total number of points that can be scored is 60, as specified alongside the questions. The final grade equals $1 + 9 * \frac{points}{60}$, rounded to the nearest half grade (except for grades between 5 and 6 which are rounded to nearest full grade).

Good luck!

ALTERNATIVE 1: Exam for the current academic year (2017-2018)

Answer the questions in this version of the exam if you followed the course this academic year (with lectures by Joeri de Ruiter and Jaap-Henk Hoepman).

Question 1 (9 points): BGP

a. *(3 points)* Two security extension for BGP that are currently standardised in RFCs are origin authentication (using ROAs) and path validation (using BGPsec). What attacks does path validation protect against, which origin authentication does not?

b. *(3 points)* As we see over and over again, adoption of new protocols on the Internet typically goes very slow and this will probably also be the case for BGPsec. Can we already benefit from BGPsec if it is only partly deployed (i.e. some autonomous systems support it and some not)? Explain why.

c. *(3 points)* In order to get around many of the legacy issues, SCION introduces a new architecture for the Internet. For the discovery of routes, they use a similar approach as BGPsec, as every AS along the path signs the PCB (Path-segment Construction Beacon) so it can be checked for authenticity. Unlike BGPsec, every AS also adds a so-called *hop field*. Explain how this field is used in the data plane.

Answer:

a. Announcing 'fake' paths, i.e. pretending to be able to route traffic to a victim AS, in order to intercept the traffic.

b. Yes, paths can already be verified if all autonomous systems on this path support BGPsec. This can then be used in the policy to determine routes. If there is only one AS on a path that does not support BGPsec, the path cannot be verified.

c. The HF contains information for a specific AS with authenticated instructions how a packet should be routed. A sender defines how a packet should travel by including the corresponding HFs in the header of the packet. Every AS will then authenticate its dedicated HF, and use this to route the packet to the next AS.

Question 2 (9 points): GSM

a. *(3 points)* On GSM networks it is possible to perform an active man-in-the-middle attack. Why is this not possible (or less trivial) on later generations of the standard such as UMTS and LTE?

b. *(3 points)* A telecom provider is considering updating its cell towers to only support the more secure A5/3 cipher. Assuming we cannot break A5/3, would this measure protect against active man-in-the-middle attacks? Explain why.

c. *(3 points)* Currently it is quite easy to perform IMSI-catching attacks, as a phone will always reveal its IMSI when asked for it. This can, for example, be used to trace users. To counter this, a provider suggests the following solution: a phone will encrypt its IMSI before sending it with the public key of the home network. Will this solution protect against all traceability attacks (both for GSM and 3G)? Explain your answer.

Answer:

a. GSM does not provide mutual authentication, so a mobile phone does not know whether it is connected to the genuine network. UMTS and LTE do provide mutual authentication making it harder to set up a rogue base station.

b. No, if the mobile phones still support GSM and A5/2, a MitM can crack the key in real-time and use this for authentication against the real network, after tricking the victim to connect to its fake base station. This is possible because the same key is used for all ciphers.

c. No, for example, the phone still needs to provide the identity of its home network so the network knows where to retrieve the necessary information. If a phone is the only one in a network that is from a specific provider it is trivial to trace it. Another example is when you don't randomise the encryption and always send the same ciphertext, which can then be used to trace the user. Yet another example is the linkability attack by Arapinis et al. in 3G. This attack does however not work in GSM, as there is nothing for the phone to verify which could lead to different error messages.

Question 3 (9 points): WiFi

a. *(3 points)* A restaurant offers WiFi and decides to protect the traffic using WPA2-Personal (using CCMP). The password for this is printed on the menu so all customers can make use of the free WiFi. Are the customers protected against passive eavesdropping attacks? Explain why.

b. *(3 points)* With KRACK (Key Reinstallation Attacks) we have seen that nonce reuse can lead to serious security problems. Explain how nonce reuse could lead to the recovery of encrypted data.

c. *(3 points)* If you could make a change to the cryptography that is used in WiFi, what change would prevent this attack (i.e. the one in .b)?

Answer:

a. No, if a passive attacker can observe the 4-way handshake between a victim and the access point, it can observe the nonces that are exchanged. Since the password is public, the attacker has enough information to compute the PTK. The PTK can then be used to decrypt all the information exchanged between the victim and the access point.

b. The nonce is used to generate the keystream, by reusing the nonce the same keystream is used to encrypt two (different) packets. Assume one of those packets contains known plaintext and the other packet the data we want to learn. Compute the xor of the encrypted packet containing the known plaintext with the known plaintext. This will result in the keystream. Now the unknown data can be retrieved by computing the xor of the keystream and the encrypted data packet.

c. For example: use a block cipher instead of a stream cipher. This way the xor of the known plaintext and the encrypted plaintext will not result in the original keystream.

Question 4 (3 points): Consider the following scenario. A new botnet scans for vulnerable hosts. Potential vulnerable hosts are reported and another bot will try to exploit these. If a host can be successfully exploited, the bot software is uploaded, installed and configured and finally the bot connects back to a C&C server to wait for commands. What patterns (low/medium/high) do you expect to see in the netflow data for the different phases with regard to number of flows and packets per flow? Explain why.

Answer:

- Port scan: high number of flows with low PPF (all hosts are scanned, but checking for an open port requires little traffic per flow)
- Executing the exploit: medium number of flows with medium PPF (only machines with the correct port are connected to and exploit requires some more traffic to be executed)
- Installation of the bot software: low number of flows with high PPF (only connection to vulnerable machines and executable causes higher amount of traffic)

- Connection to the C&C server: low number of flows with medium PPF (only infected machines are connected to and traffic is medium as bots are waiting for commands)

Question 5 (15 points): On distributed algorithms

a) *(5 points)* Will the following program (eventually) terminate, or is it possible that it runs forever? Assume that reading or writing a single variable is atomic.

$a = 1$
$b = 1$

thread **while** $a \neq 0$
do $b \leftarrow (b + a) \bmod 2$

thread **while** $b \neq 0$
do $a \leftarrow a + 1$
$a \leftarrow 0$

b) *(3 points)* What are the three properties a mutual exclusion protocol must satisfy?

c) *(7 points)* Consider two processors 0 and 1 that can both read and write one global atomic variable p, initially 0. Why is the following protocol (code for processor i) not a mutual exclusion protocol?

while *true*
do while $p \neq i$ **do** */* wait */*
/ critical section */*
$p \leftarrow 1 - p$

Answer:

a) Suppose $b = 1$.

Let the second thread take a single step (changing a from odd to even or vice versa). If a has become odd, let the first thread take two steps. Then after those two steps b again is equal to 1. If a has become even, let the first thread take one step. After that single step b still equals 1.

This can be repeated forever (assuming a is unbounded).

Alternative answer: Let second thread do one step; now a is even. The continue with the following steps forever: let the first thread do one step (because a is even, b remains 1), and then let the second thread do two steps so a becomes even again.

P.S.: Note that the assignment $a \leftarrow 0$ is *not* part of the loop body!

b) A mutual exclusion protocol needs to satisfy the following three properties: *Mutual exclusion*: there is at most one processor in the critical section, *Progress*: if there is at least one processor enters, and the critical section is empty, then one of these processors will eventually get access to the critical section, and *No starvation*: if a processor enters, and if all processors that get access to the critical section release it, then it will eventually get access

c) This protocol does not satisfy progress: because $p = 0$ initially, if only processor 1 ever wants to enter, it is blocked forever (while it should be able to get the critical section if no other processor is contending, because of the no-starvation condition).

Question 6 (15 points): This question considers self-stabilisation.

a. *(2 points)* What is the difference between a central daemon and a distributed daemon?

b. *(3 points)* Which of these two daemons is the hardest model to design a selfstabilising system for? Explain your answer.

c. *(10 points)* Dijkstra's selfstabilising mutual exclusion protocol (as explained in class) runs on nodes numbered 0 to N. The state of each node is given as a value in $\{0,\ldots,K-1\}$. The system stabilises under the central daemon when $K \geq N$. What goes wrong when $K < N$?

Answer:

a. Under the central daemon exactly one node takes a step at a time. Under the distributed daemon at least one node, put possibly more nodes, take a step simultaneously.

b. The distributed daemon is the hardest model to design a selfstabilising system for. For one, the central daemon is a special case of the distributed daemon (so every protocol that self stabilises under the distributed daemon also self-stabilises under the central daemon). Intuitively, the distributed daemon is harder because if a node takes a step, considering all other states it has access to, to move closer to the desired, legitimate, states, then another node may update its state simultaneously. As both nodes are unaware of their 'moves', they may 'move' in opposite directions.

c. Take the case where $N = 3$ and $K = 2$. The possible states are 0 and 1. Let the initial state be 0101 (i.e. node 0 has value 0, node 1 has value 1, node 2 has value 0 and node 3 has value 1. Consider the following sequence of steps:

   - Node 3 takes a step. New state 0100.
   - Node 2 takes a step. New state 0110.
   - Node 1 takes a step. New state 0010.
   - Node 0 takes a step. New state 1010.
   - Node 3 takes a step. New state 1011.
   - Node 2 takes a step. New state 1001.
   - Node 1 takes a step. New state 1101.
   - Node 0 takes a step. New state 0101.

   We reached the state we started in. This state is not legitimate (all nodes are enabled), so the system does not converge.

END OF EXAM ALTERNATIVE 1

ALTERNATIVE 2: (Resit) Exam for the previous academic years

Answer the questions in this version of the exam if you followed the course the previous academic years (with lectures only by Jaap-Henk Hoepman).

Question 1 (15 points):

a) *(5 points)* Will the following program (eventually) terminate, or is it possible that it runs forever? Assume that reading or writing a single variable is atomic.

$a = 1$
$b = 1$

thread **while** $a \neq 0$
do $b \leftarrow (b + a) \bmod 2$

thread **while** $b \neq 0$
do $a \leftarrow a + 1$
$a \leftarrow 0$

b) *(3 points)* What are the three properties a mutual exclusion protocol must satisfy?

c) *(7 points)* Consider two processors 0 and 1 that can both read and write one global atomic variable p, initially 0. Why is the following protocol (code for processor i) not a mutual exclusion protocol?

while *true*
do while $p \neq i$ **do** */* wait */*
/ critical section */*
$p \leftarrow 1 - p$

Answer:

a) Suppose $b = 1$.

Let the second thread take a single step (changing a from odd to even or vice versa). If a has become odd, let the first thread take two steps. Then after those two steps b again is equal to 1. If a has become even, let the first thread take one step. After that single step b still equals 1.

This can be repeated forever (assuming a is unbounded).

Alternative answer: Let second thread do one step; now a is even. The continue with the following steps forever: let the first thread do one step (because a is even, b remains 1), and then let the second thread do two steps so a becomes even again.

P.S.: Note that the assignment $a \leftarrow 0$ is *not* part of the loop body!

b) A mutual exclusion protocol needs to satisfy the following three properties: *Mutual exclusion*: there is at most one processor in the critical section, *Progress*: if there is at least one processor enters, and the critical section is empty, then one of these processors will eventually get access to the critical section, and *No starvation*: if a processor enters, and if all processors that get access to the critical section release it, then it will eventually get access

c) This protocol does not satisfy progress: because $p = 0$ initially, if only processor 1 ever wants to enter, it is blocked forever (while it should be able to get the critical section if no other processor is contending, because of the no-starvation condition).

Question 2 (15 points): This question considers self-stabilisation.

a. *(2 points)* What is the difference between a central daemon and a distributed daemon?

b. *(3 points)* Which of these two daemons is the hardest model to design a selfstabilising system for? Explain your answer.

c. *(10 points)* Dijkstra's selfstabilising mutual exclusion protocol (as explained in class) runs on nodes numbered 0 to N. The state of each node is given as a value in $\{0,\ldots,K-1\}$. The system stabilises under the central daemon when $K \geq N$. What goes wrong when $K < N$?

Answer:

a. Under the central daemon exactly one node takes a step at a time. Under the distributed daemon at least one node, put possibly more nodes, take a step simultaneously.

b. The distributed daemon is the hardest model to design a selfstabilising system for. For one, the central daemon is a special case of the distributed daemon (so every protocol that self stabilises under the distributed daemon also self-stabilises under the central daemon). Intuitively, the distributed daemon is harder because if a node takes a step, considering all other states it has access to, to move closer to the desired, legitimate, states, then another node may update its state simultaneously. As both nodes are unaware of their 'moves', they may 'move' in opposite directions.

c. Take the case where $N = 3$ and $K = 2$. The possible states are 0 and 1. Let the initial state be 0101 (i.e. node 0 has value 0, node 1 has value 1, node 2 has value 0 and node 3 has value 1. Consider the following sequence of steps:

   - Node 3 takes a step. New state 0100.
   - Node 2 takes a step. New state 0110.
   - Node 1 takes a step. New state 0010.
   - Node 0 takes a step. New state 1010.
   - Node 3 takes a step. New state 1011.
   - Node 2 takes a step. New state 1001.
   - Node 1 takes a step. New state 1101.
   - Node 0 takes a step. New state 0101.

   We reached the state we started in. This state is not legitimate (all nodes are enabled), so the system does not converge.

Question 3 (15 points): This question considers bitcoin/blockchains.

a. *(4 points)* Explain how mining a block in the bitcoin blockchain works.

b. *(4 points)* Why would miners resist a change in the mining function, e.g. when moving from proof-of-work to proof-of-stake?

c. *(7 points)* Explain how transactions in the bitcoin network can be de-anonymised.

Answer:

a. In the bitcoin blockchain, mining a block requires one to find a nonce n such that the hash h of the current block contents b (represented by the hash of a the Merkle tree containing all transactions in the block) together with the nonce is less than a target value t, i.e. $h(b;n) < t$.

b. Miners would resist a fundamental change in the mining function as it would render all their investments in specialised hardware useless.

c. The bitcoin blockchain is public. From this a *transaction graph* (linking all transactions by looking at which inputs consume which outputs) can be constructed. From this transaction graph an *address graph* can be constructed linking bitcoin addresses. These addresses are anonymous and typically transactions use fresh output addresses. Yet using *heuristics* (like the assumptions that all inputs to a transaction belong to the same user, and that the second output of a transaction is a 'change' address and hence belongs to the same user as the input addresses) addresses can be grouped to belong to the same entity/user. Once one address in this set is linked to user identity (e.g. when exchanged for goods or other currencies), anonymity of all addresses in the set is destroyed.

Question 4 (15 points): This question is related to mix networking. In this context, answer the following three subquestions.

a. *(3 points)* What are anonymous return addresses?

b. *(3 points)* Why are they necessary?

c. *(9 points)* How do they work?

Answer:

a. An anonymous return address is a datastructure attached to a message that specifies a route for responding to that message, together with intermediate keys necessary to encrypt and thus secure the reply.

b. Anonymous return addresses are necessary to allow a recipient of a message to reply to it without learning who the recipient of the reply (i.e. the sender of the original message) is.

c. Concretely, an anonymous return address is the triple

$$(\{S_1; M_2; \{\cdots \{S_k; A\}_k \cdots\}_2\}_1, M_1, K_A)$$

where K_A is a fresh public key generated by the recipient A, where $M_1, \ldots, M_k$ are the names of the intermediate mixes, where M_i can decrypt $\{x\}_i$, and where $S_1, \ldots, S_k$ are fresh symmetric keys generated by A. To use it, the sender of the reply encrypts the reply using K_A and sends it to mix M_1, together with the first part of the anonymous reply block. Each mix i in the path, when receiving an encrypted reply m and part of the anonymous reply block $\{S_i; M_{i+1}; \{\cdots \{S_k; A\}_k \cdots\}_i$ decrypts the reply block using its private key k_i to obtain S_i and M_{i+1} and then encrypts m with S_i and passes it on to M_{i+1} together with the remainder of the reply block (i.e. $\{S_{i+1}; M_{i+2}; \{\cdots \{S_k; A\}_k \cdots\}_{i+1}$).

END OF EXAM ALTERNATIVE 2