UMEÅ UNIVERSITY, DEPARTMENT OF COMPUTING SCIENCE

Written Assignment 1 (20 points)

Distributed Systems - 5DV147

You need to complete and submit this assignment individually. Collaboration is permitted only as described in the Guidelines for compulsory assignments and the Honor code. You need to submit this assignment by November 29 at 13:00 (hard deadline). You can either email your solution to us (5dv147-staff(at)cs.umu.se) or drop off your solution in the box outside the department.

1 SECURITY (8 POINTS)

The Transport Layer Security (TLS) protocol is the most widely employed protocol to secure communication on the Internet. However, it has become a victim of its own popularity and many vulnerabilities have been found over time, either in the TLS specification itself, or in an implementation of it. For example, during the lecture you saw that an early version of the Netscape browser failed to properly initialize the pseudo-random number generator, which lead to the creation of predictable session keys. Such examples serve as a lesson about how to design secure protocols and systems. As the saying goes, "those who fail to learn from history are doomed to repeat it."

Unfortunately, during the security lecture we have only been able to briefly cover how to build secure distributed systems. Therefore, to further your knowledge, we now ask you to **document one vulnerability** that was found in TLS or the way it was implemented. First, choose a vulnerability of TLS. You can do this either by reading the Wikipedia[1] article, or searching through popular vulnerability databases[2]. Make sure you read in-depth, i.e., you

[1] https://en.wikipedia.org/wiki/Transport_Layer_Security#Attacks_against_TLS.2FSSL

[2] http://cve.mitre.org/find/index.html

might need to look up terms such as “replay attack” or “man-in-the-middle”, or dig deeper into the RSA algorithm. Once you think you have enough information, write a report of **at least 1 page but no more than 2 pages** (Times 12pt, 2cm-margin on all sides). You are free to structure the report any way you feel fit, however, you should make sure that it is as informative as possible. Write it for a person that has just attended the security lecture, i.e., is familiar with security in general and the workings of TLS, but does not have in-depth knowledge about building secure protocols. We would like to see the following issues covered:

- Some background and context (What part of TLS? What implementation? What was it supposed to do?)
- How was the vulnerability discovered?
- What does the vulnerability allow? What basic attack (e.g., replay) is it based on?
- What is the potential impact? Can an exploit be easily made? What users are affected?
- How easy is it to remedy the vulnerability? Is there a temporary workaround? Does one have to do a software update? Must the protocol be redesigned? Would this affect compatibility with “old” systems?
- What would you recommend to prevent such a vulnerability in the future? What have you learned by reading about this vulnerability?

As computing scientists, we realize that it is a lot more fun to write code instead of English. However, by having to write, instead of just read about a vulnerability, you not only improve your English writing skills, but also your communication and synthesis skills in general. Plus, as the saying goes, “you never really understand something unless you can explain it”.

To check whether the report is informative enough, you may ask a colleague to read it and see how much he/she learned out of it. Note, however, that the writing itself must be **individual**.

2 Publish-Subscribe Systems (8 points)

Publish-subscribe systems, sometimes called *distributed event-based systems*, are used to disseminate information (or events) to multiple recipients through an intermediary. In these systems, a large number of publishers (producers) publish structured events to an event service and a large number of subscribers (consumers) express interest in particular events through subscriptions which can be arbitrary patterns over the structured events.

There are many examples of such systems. For example, if you want to fly to certain destination, let’s say Melbourne, you may register on certain services and get notified when the

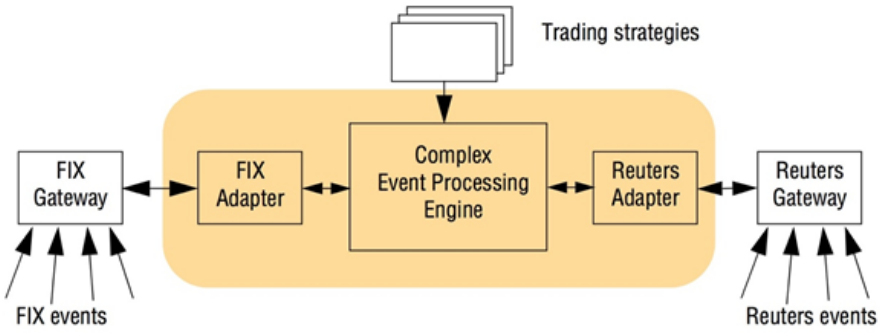
ticket price drops below a point, let's say 5K SEK. Another example that may be very familiar to you is the Really Simple Syndication (RSS) that is used to publish information that is frequently updated such as news feeds.

Let's imagine that you are hired to design a basic financial trading system like the one presented in section 1.2.3 of the course book, you can see the figure illustrating the system below. You quickly realized that this type of system perfectly matches the publish-subscribe pattern learned in class and chose to designed the system according to that pattern. We now ask you to show us your design.

In your design it is important to cover at least the following aspects:

- The architecture, centralized or decentralized?
- Types of subscription filters to use (channel-based, topic-based, content-based, or type-based), and how would the system be able to filter events accordingly.
- How publishers and subscribers would register to the system and how would that information be maintained.
- How events are sent from publishers to subscribers.
- How to assure a level of confidentiality so that subscribers are only able to see events sent to them.
- How would you make the system tolerant to crash failures?

Figure 2.1: An example financial trading system, from [DS5]



[DS5] Coulouris G., Dollimore J., Kindberg T. and Blair G.: *Distributed Systems - Concept and Design.* Fifth edition. Addison Wesley (2005)

3 System Performance (4 points)

As we saw in class, high level requests such as loading a web page, fetching a data file from the network, or querying a database, depend on a number of low level operations. Each

Operation	Latency (ns)
L1 cache reference	0.5
Branch mis-predict	5
L2 cache reference	7
Mutex lock/unlock	25
Main memory reference	100
Compress 1K bytes with Zippy	3,000
Send 2K bytes over 1 Gbps network	20,000
Read 1 MB sequentially from memory	250,000
Round trip within same datacenter	500,000
Disk seek	10,000,000
Read 1 MB sequentially from disk	20,000,000
Send packet CA->Netherlands->CA	150,000,000

Table 3.1: Latency numbers every programmer should know.

low level operation adds a little bit of latency to the overall high level request latency.

When you program web applications, it is very important that requests are served with very short latencies . If you were to design an on-line store, you would lose many customers if it takes a minute to check whether certain product is available. Thus, it is very important for you to understand where the time is spent so that you can optimize at the right places.

Taking into account the latencies presented in class[3] and reproduced above[4], propose 3 strategies that you could use to reduce latency in a web application that you (theoretically) have developed.

[3] From "Latency numbers every programmer should know", Jeff Dean

[4] You can also take a look at http://www.eecs.berkeley.edu/~rcs/research/interactive_latency.html if you want to know how those numbers have varied in the past (and into the future).