DSV SU/KTH | ***IV2018(part 2)/Säk1b/IV2024***

Information and Computer Security

Autumn Term 2008

Alan Davidson

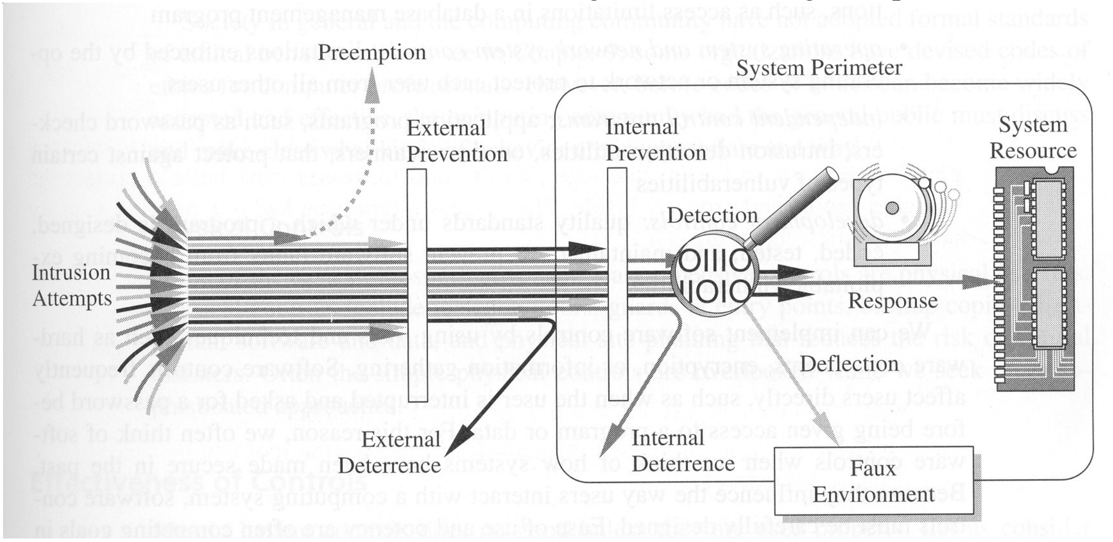
Suggested Solutions to the Exam 2008-10-22 and Comments on the Marking

Note that since these answers can cover several aspects of possible answers and also discuss these answers, they can contain more than would normally be required for full credit. Indeed, it is normally a strength if the student can give concise exam answers, whereas the following text seeks to be a thorough, comprehensive discussion.

Problem 1

One possible measure to protect an IT system is an Intrusion Detection System. Suggest and motivate three other general, effective measures that a systems administrator can use to protect an IT system before the stage at which an IDS becomes necessary. Include at least one non-technical measure. Good answers will show a breadth of possible measures.

At the first lecture we saw an illustration from Pfleeger's book [Pfleeger07, p25]:



We see that before the detection phase (in the question represented by IDS) we have several points at which earlier controls could be introduced. We could generalise these to be *pre-emption*, *prevention* and *deterrence*. Since the question asked for breadth, a good answer might suitably present protection measures from each of these classes.

A pre-emption method (and a non technical one) might by to publish information to prospective attackers that can dissuade them. It might be an explanation that nothing of value is kept within the system, or convincing information that anyone found responsible for an intrusion can expect do be dealt with harshly (an equivalent from the physical world is posting a “Shop-lifters will be prosecuted” sign in a shop). Some students pointed out how such messages might instead act as a challenge and an enticement to some attackers. This may be true, but since no method is perfect, a case can nevertheless be made that this could be an effective method.

Whether a measure should be classified as prevention or deterrence may not always clear. I suggest that measures that clearly meet the threat of an attack are prevention methods, whereas measures that lessen the effectiveness of an attack are deterrence methods. If prevention is concerned with understanding attacks and countering them, deterrence is about understanding the value of resources and making it difficult to compromise.

An effective prevention method can be a firewall. Intrusion attempts can come from networks and many times will be follow a pattern that differs from normal network traffic. Blocking traffic that is not part of normal network communication with an organisations servers will prevent such attacks.

An effective deterrence measure is to implement a proxy server at a point on the network that is to some degree on the outer limits of a local network. In this way network traffic that is external to an organisation, and therefore less likely to be trusted, is directed to a server that both contains less valuable resources and is in a part of the network that is less sensitive.

Some answers brought up faux environments such as honey pots as possible measures. It is however very difficult to see how an intrusion attempt can be deflected to such an environment before it has been detected as an intrusion.

A number of students chose to answer with more than the three measure that the question asks for. I should perhaps therefore emphasise that answering with more than the question asks for is always a worse answer. Apart from anything else, it is indicative of a lack of understanding of the subject and of your own knowledge. It also creates unnecessary extra work for an examiner (assuming that the answer is not rejected immediately as an improper answer to the question, which it maybe should be). The only examination strategy that I can imagine is fair and just in such a situation is to pick the three worst answers and mark according to them. Rather than trying to convince students that there are very many arguments for them not to start making assumptions outside of the framework of a question, I would like to just remind students that it is always a good idea to (simply) answer the question as it is written.

Several students seemed to take this question as an opportunity to write pages and pages on things that they knew details about, such as details of choosing passwords, or Saltzer and Schreoder's principles for the design of security software. Such off-subject dumping of knowledge never improves an answer. It does annoy the examiner that there is so much irrelevant text to wade through when marking exams! Since it definitely reduces readability, there is a case to be made for not marking such answers at all.

Problem 2

> The concept of key-space for an cryptographic algorithm is similar to that of the space of all possible passwords that an authentication method allows.
>
> a) Identify and motivate a factor (or factors) that effects the strength of keys and of passwords which are similar for both of these concepts.
>
> b) Identify and explain a factor (or factors) of the space of all possible passwords that can make the authentication method weak, and that you would not normally expect to see occurring in the key space for a cryptographic algorithm.

a) Length is surely the easiest factor to cite. The basic space of all possible passwords is a function of the number of characters that may be used in a password and the number of positions of such characters, i.e. the length. Modern symmetric cryptographic methods in support of IT do not share same factor of the number of possible characters in that they are normally binary numbers, i.e. there are only two "characters" to choose between, *1* and *0*. The size of the key space, and thereby the strength, is however similarly effected by the length of the key.

There are some subtle points of similarity and difference between password length and key length that could suitably be ignored for the sake of the exam question, but which some exam answers went into. Passwords will normally have variable length, meaning that the basic space of all possible passwords will be a product of all the possible (*character set size* raised to the power *number of positions*) from e.g. *number of positions = 0* to *number of positions = 256.* Modern *c*ryptographic keys, on the other hand, tend to have a fixed length. DES for example has an effective key length of 56. Some encryption algorithms do allow variable length keys.

b) One possible answer: Cryptographic keys are generated by mathematical functions that are designed to give close to random selection of keys. Within the space of possible keys, these mathematical functions ensure that any one key is as likely to occur as another. Passwords on the other hand are most commonly chosen by people, not by mathematical functions. A normal requirement for passwords is that the creator be able to remember them. For this reason people will tend to limit their choices to patterns of characters that are easier for themselves, or for people in general to remember. For this reason, some passwords within the space of all possible passwords will be more likely than others. The space therefore becomes a topology rather than an even space. This is why dictionary attacks can be assumed to be more affective than brute force attacks for passwords. There is no equivalent to dictionary attacks for guessing at cryptographic keys.

Problem 3

Describe each of the following IT security related terms. Also, clearly relate each of your descriptions to a closely connected IT security concept of your own choosing, and give an example of an application of these tools/threats/concepts:

- Steganography
- Risk Analysis
- Formal Verification
- Common Criteria

This question is of a kind that checks knowledge of some of the more detailed terms and concepts that are part of the course material. When marking the answers I assumed that being able to provide good descriptions, comparisons and examples of at least three of these terms during the exam could be regarded as fulfilment of the course goals and therefore worthy of a pass grade.

Steganography

Description - Steganography is the name for a class of methods that are used to hide the existence of information.

Comparison – The word c*ryptography*(in direct translation *hidden writing*) has come to mean the art and science of hiding the content of a collection of information (often a message), i.e. the fact the information exists is not hidden, but the information itself is difficult to extract. Cryptographic messages can therefore be subject to traffic analysis techniques, i.e. deriving information from the existence of traffic even if the content is not possible to interpret. This is in contrast to *steganography* (directly translated *covered writing*), here the fact that a message exists is hidden.

Example – It is possible to hide information within jpeg encoded pictures. This method is known to have been used by terrorist to spread text messages within seemingly innocuous pictures.

A number of answers mixed up examples of steganography with the description. Note that it is **not** the art and science of hiding messages in pictures. That is just one of the many possible steganographic methods.

Risk Analysis

Description – To quote [Bishop05, p15]:

To determine whether an asset should be protected, and to what level, requires analysis of *the potential threats against that asset and the likelihood that they will materialize. The level of protection is a function of the probability of an attack occuring and the effects of the attack should it succeed.*

Comparison – Vulnerability analysis is the process of testing for security flaws in a system, often with the aid of automated tools. The major difference between risk analysis and vulnerability analysis is the aspect of factoring in the likelihood. Otherwise we can say that risk analysis will tend to be applied at an organisational level, whereas vulnerability analysis is an activity directed at computer systems and networks. The goal of risk analysis is to ascertain what should be protected, whereas vulnerability analysis gives an indication as to what flaws should be fixed.

Example – As part of a risk analysis, an organisation might determine that a major asset is their customer database. Losing this to competition would be a threat to the whole organisation's survival. However, since the customer database is kept in the charge of trusted personnel and on an old computer that is not connected to the Internet, and with backup on a save media that is kept well secured at another physical location, the likelihood that the customer database will be lost or discovered by competitors will be very small, and any more protection measures would be superfluous.

Formal Verification

Description - Method by which the semantics of the design of a system is expressed with mathematical stringency, and thereby the design and implementation can be verified to be correct.

Comparison – The process of testing is a hunt for flaws, but as Dijkstra has written: "Program testing can be used to show the presence of bugs, but never to show their absence" [Dijkstra72]. Formal verification can on the other hand (at least in theory) prove that there are no flaws in the implementation of a design. With test one can put as much effort into the process as one deems suitable for the level of assurance required for the situation. Formal verification is regarded as an expensive measure that is most suitably applied in situations where the very highest levels of assurance are required.

Example – The reference monitor of a trusted operating system is a small but vital component. It must mediate all accesses to memory, and if it can be subverted then the security of the system overall will be in question. By specifying the design of the reference monitor in terms of predicate calculus one can symbolically manipulate that design in order to prove its completeness, i.e. that the process can not be subverted. A static analysis of the semantics of the implemented code can be compared to that design to show the presence or absence of bugs even before the code has ever been run.

Common Criteria

Description - The Common Criteria is an internationally standardised method by which the security functionality of software can be formalised, assessed, and certified.

Comparison – The Trusted Computer System Evaluation Criteria - TCSEC (also know as “The Orange Book”) was a predecessor to the Common Criteria that was developed in the United States in the late '70s and early 80's. One of the main differences between the methods is that TCSEC was designed for the evaluation of secure operating systems, whereas the Common Criteria is intended as a method for evaluating software in general.

Example – An organisation that wishes to fulfil their needs for secure filtering of network traffic might formalise their requirements as a Protection Profile (PP) for a firewall, according to the Common Criteria Guidelines. They may also have this PP certified by an CC authorised third party in order to assure that the PP itself upholds the CC requirements for such a document. Such a firewall PP will normally be general enough to match the requirements of any number of prospective customers.

A software vendor who wishes to show that their product (e.g. firewall) is trustworthy may describe its security functionality in a Security Target document. They can then evaluate their product (in CC terms the Target of Evaluation – TOE) in relationship to the Security Target, or rather have it evaluated by an authorised third party, and even have their product certified as CC compliant. Especially if the ST was derived from an existing PP, the product can therefore be shown to meet customer requirements. The CC also specifies 7 different Evaluation Assurance Levels (EALs) which define at what level of confidence an evaluation can say that the security functionality requirements have been met. Note that an EAL does not specify a level of security so much as an level of assurance that the ST stated security functionality requirements for a piece of software have been fulfilled.

Note that the Common Criteria is not intended as an evaluation method for complete software or computer systems, but only for individual software.

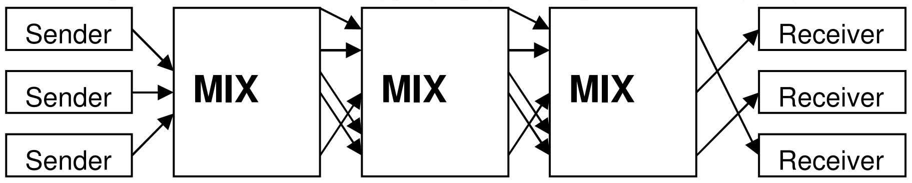
General comments...

Many students entirely missed the requirement for a related concept and example, or did not make it clear from their presentations what concepts they were attempting to relate to or where the example began and ended. Perhaps this is because students are studying from previous suggested solutions to exams from the course before the Bologna system was introduced? In previous years the older version of this problem was less exacting, and likewise the suggested answers. It is therefore important to note that of late the format of this question has changed. To emphasise the newer requirements, I have attempted to be extra clear in the above suggested answer. Once again the importance of being sure that students answer the question as written is emphasised.

“Relate each of your descriptions to a closely connected IT security concept of your own choosing” does not mean that you can simply state that another concept is related. You must of course explain **how** they are related too.

Problem 4

The diagram below is illustrates the basic architecture of a privacy enhanced email system. Explain why it is an advantage to have a sequence of several mix nodes between sender and receiver. Explain furthermore how cryptography can improve the anonymity service.



In very general terms, we can understand that having several mix nodes implements greater defence in depth. If the security of one mix node should fail then we have assurance in that the remaining nodes will be sufficient to uphold the anonymity service. Some of the possible ways in which a single node might fail are:

- A system might be subverted, and record the IP addresses of the sender together with the recipient. If there were only one node then this information would be enough to undermine the anonymity service. If however either the sender or recipient were another mix node in a chain of nodes, then all of those nodes would have to be subverted in order for the anonymity service to be undermined.
- By eavesdropping both incoming and outgoing traffic to a node it may be possible to match factors such as the delivery times and message sizes, and thereby match the sender with the

recipient. In order to accomplish this match with a sequence of mix nodes it would be necessary to either eavesdrop incoming and outgoing traffic at all of the involved nodes, or to have to guess at who the sender and recipient will be and eavesdrop immediately in their vicinity. With a mix of nodes, the route that a message will take will not be known a priori, and eavesdropping will be all the more difficult.

- If a node were a pseudonymous remailer (not usual in *mixnets*, but conceivable), that means it would keep a table of the senders' addresses mapped to connected identities. Messages would be stripped of any information pertaining to the true sender and re-sent with an anonymous identity. One advantage is that it is simple in such a system to allow for replies to anonymous messages by forwarding them through the same server and swapping back anonymous identities for true addresses before forwarding. However, a single server that contains such tables might be forced to reveal their associations by national authorities, as was the case with a classic anonymity server at penet.fi (see e.g. http://en.wikipedia.org/wiki/Penet_remailer) and the reason for it discontinuing the service. If a chain of servers were spread geographically, then the international cooperation that it would require to have all the necessary associations revealed will make the process all the more difficult.

Given the threats from eavesdropping both within a node and in the traffic to and from a node, a good anonymity service must not allow enough information to be gleaned from a message to derive the link between sender and receiver at any point in its delivery. If the message were sent in clear then all of that information would be available in the message as it comes from the sender. It is therefore necessary to hide not only the recipient of a message, but also any of its content that could be matched in two versions of that message eavesdropped en route.

With the help of asymmetric cryptosystems one can encrypt a message with aid of the public key of a mix node, knowing that only that node will be able to decrypt the content. The recipient of the message in the next step from that node can also be encrypted with the aid of said public key. In this way, only the original sender and the receiving node will ever know who is the next recipient in the chain, as it is encrypted for all other parties. Now by successively asymmetrically encrypting the recipient and the message content for each node in a chain, each node will unveil the message for the next step by decrypting the message it receives. Traceable links between the message as sent and as delivered are effectively masked.

An added bonus of the asymmetrical cryptography scheme is that only the legitimate recipient node will be able to decrypt their layer of the message. So long as the public keys that are used are legitimate, anyone who attempts to spoof a mix node to thereby attack anonymity will not be able to.

Problem 5

> Pick three of Saltzer and Schroeder's eight principles for the design and implementation of security mechanisms, and explain how failure to observe each of these principles can result in three classes of common program vulnerabilities.

This question does not ask for any old vulnerability, but *common program vulnerabilities.* A good starting point therefore would be to pick program vulnerabilities that can be motivated as common. Here we take the three most common vulnerabilities from NISTs statistics over reported vulnerabilities thus far during 2008 (see http://nvd.nist.gov/statistics.cfm). Students are not necessarily expected to have these statistics at hand for the exam, but luckily enough these common vulnerabilities were among those discussed at lectures.

The principle of complete mediation requires that all accesses to objects must be checked to ensure that they are allowed [Bishop05, p203]. SQL Injection is a common program vulnerability that can occur when input taken from a client is assumed to be well formed and catinated into a SQL query. Carefully crafted input can adapt the query to result in database accesses that were not intended by the programmer. We can put this down to the fact that the access to the database was not properly checked to ensure that it was allowed. Carefully checking the client side input to make sure that it is properly formed and therfore allowed will alleviate the vulnerability.

Short notes - The same principle can be linked to several other of the most common vulnerabilities: Cross-Site Scripting, buffer errors, input validation, path traversal, code injection. The fact that so many of the common vulnerabilties can be linked to this principle suggests that it should be part of a good answer. Another principle that can be linked to SQL Injection is that of fail-safe defaults. Following this principle one would not allow the input to be part of the executed statement, but intead only use pre-constructed SQL statements.

The principle of least common mechanism states that mechanisms used to access objects should not be shared [Bishop05, p206]. One common cause of buffer overflow vulnerabilities is that languages such as C and assembler have mechanisms for accessing memory that are all too easily shared. Vulnerabilities can occur when parts of memory can be overwritten by user input (assisted in part by not keeping to the principle of complete mediation - see above) and thereby effect the course of execution in ways that the programmer had not predicted. Many other languages such as Java and Lisp have automatic memory management which means that mechanisms that share memory are not shared and not susceptable to the same kinds of buffer overlow vulnerabilities.

The principle of least privilege states that a subject should be given only those privileges that it needs in order to complete its task [Bishop05, p201]. NIST has the class of vulnerabilities *Permissions, Privileges, and Access Control*, which covers flaws from improperly handling among other things, privileges. There would seem to be a direct mapping from the principle to the vulnerability here. Giving programs more priviliges than they need opens up the possibility for such programs to be manipulated in order to subvert access control mechanisms.

There is some unintentional leeway for interpretation in the question. Clearly it is about three of Slatzer and Schroeder's principles, and the intention was that each principle should be matched to one class of vulnerability. However, one can interpret the mapping to be to up to 9 vulnerability classes in total (depending on how many of the classes might be duplicated for differing principles). All such answers were therefore accepted. Credit was given for the quality of the argument rather than for the number of associations.

Note that Open Design does not say that the many eyes principle is a good way to save your code from vulnerabilities, but that the security of a mechanism should not be dependent on its secrecy. In discussion during the course we have drawn parallels to this principle and the idea of many eyes, but one should be careful about what the principal actually states.

References

Bishop05	Matt Bishop, *Introduction to Computer Security*, Addison Wesley, 2005.
Dijkstra72	E. W. Dijkstra, "Notes on Structured Programming," *Structured Programming*, O.-J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Editors, Academic Press, 1972, pp. 1-82.
Pfleeger07	Pfleeger, C. P. and Pfleeger S. L., *Security in Computing, Fourth Ed.*, Prentice Hall, 2007