COMP4108 Final Exam Practice

William Findlay

December 16, 2019

Contents

1 Preamble

1.1 Textbook

- here is a link to "Tools and Jewels"

1.2 General

1. Please follow the provided format
2. We should prioritize the mock exam over notes

Part I

Mock Exam

1 Basic Concepts and Principles

1. Provide definitions for the following:

   a) Confidentiality

   b) Data integrity

   c) Authentication

   d) Authorization

   e) Availability

f) Accountability

2. Briefly explain how repudiation violates accountability.

3. Describe the difference between a *trusted* and *trustworthy* actor.

4. Compare and contrast *privacy*, *protection*, and *anonymity*.

5. Come up with a simple example of a security policy for a house and describe a way it might be violated.

6. Label each number in Figure 1.1 using the following terms:

a) target asset

b) vulnerability

c) attacker

d) attack vector

e) threat agent

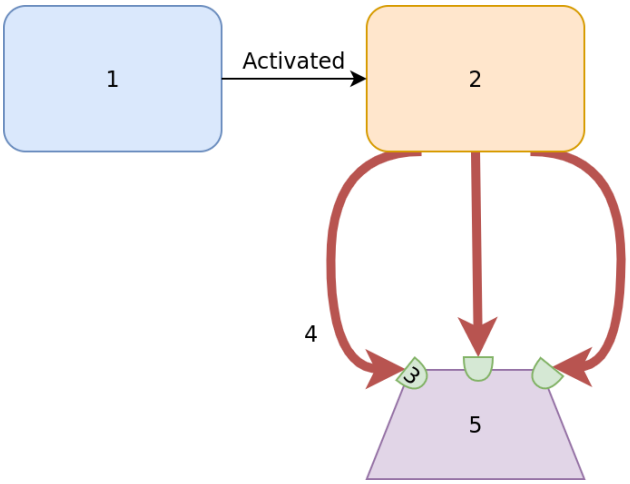


Figure 1.1

7. Draw a state machine diagram of a system's transition from a secure state to either a secure state or an insecure state.

8. Compare and contrast quantitative and qualitative risk assessment. Consider the advantages and disadvantages of each, as well as how each might work in theory/practice.

Qualitative	Quantitative

9. Consider $R = T \times V \times C$.

   a) What is this equation for?

   b) Describe each variable in this equation. How does each variable relate to the equation's purpose?

   c) Which two variables may be combined into P? What does the simplified equation look like? What does P represent?

10. Describe two risk assessment challenges.

11. Which of the following is not an adversary attribute?

    a) objectives

    b) outsider/insider

    c) methods

    d) funding level

    e) capabilities

    f) attack vector

12. What is a categorical schema? How is it different from a capability-level schema?

13. Compare and contrast a formal security evaluation with penetration testing.

Formal Security Evaluation	**Penetration Testing**

14. What is white-box pen testing? Black-box?

15. Consider STRIDE. What does each letter stand for?
    a) S:
    b) T:
    c) R:
    d) I:
    e) D:
    f) E:

16. Draw a tree model for compromising the password to a bank account. Include at least three leaf nodes.

17. Is it possible to completely test a comprehensive (and practical) set of security mechanisms for a system? Why or why not?

18. Explain the observability (or lack thereof) of security in the context of *negative goals*.

19. Assurance in security is best described as which of the following?

a) Simple, effective

b) Difficult, partial

c) Simple, practical

d) Difficult, complete

e) None of the above

2 Cryptographic Building Blocks

20. Suppose Alice encrypts a message to Bob using $E_k(m) = c$. How does Bob decrypt the message?

21. What is an exhaustive key search? What does the attacker try to do? Is this the worst case for attacking a cryptosystem?

22. Label each of the following attacks as either an action by an *active* or a *passive* adversary. Once you have labeled the attack, describe it.

a) Known plaintext attack

b) Ciphertext only attack

c) Chosen plaintext attack

d) Chosen ciphertext attack

23. What is the main advantage of a one-time pad? Describe three disadvantages. Why are one-time pads not used?

24. What is the current standard for block ciphers?

25. Describe a situation in which we would need to use a stream cipher. Why can't you use another type of cipher?

26. What is a mode of operation used for?

27. What is one major flaw with the ECB mode of operation?

28. Draw a picture of the CBC mode of operation.

29. Draw a picture of the CTR mode of operation.

30. If Alice wants to send a message to Bob using public-key encryption, _____ is used to encrypt and _____ is used to decrypt.

a) Bob's private key, Alice's public key

b) Bob's public key, Alice's private key

c) Bob's public key, Bob's private key

d) Alice's private key, Alice's public key

e) None of the above

31. If Alice wants to send a message to Bob using a public-key signature scheme, _____ is used to sign and _____ is used to verify

a) Bob's private key, Alice's public key

b) Bob's public key, Alice's private key

c) Bob's public key, Bob's private key

d) Alice's private key, Alice's public key

e) None of the above

32. How does hybrid encryption work? What role does symmetric key encryption play? Public-key encryption?

33. What three security properties do digital signature schemes provide? To whom to they provide them?

34. What two security properties do MACs provide? To whom do they provide them?

35. What security property does a cryptographic hash provide? To whom does it provide the property?

36. Describe each of the following properties of cryptographic hash functions.

a) Preimage resistance

b) Second preimage resistance

c) Collision resistance

37. How are hash functions used for password storage and verification? What property or properties of hash functions make this a desirable use case?

38. Is it generally better to MAC then encrypt, or encrypt then MAC?

3 User Authentication

39. Describe each of the following ways to defeat password authentication. For each technique you describe, provide one way to prevent it.

a) Online guessing

b) Offline guessing

c) Defeating password recovery

d) Bypassing authentication interface

e) Password capture

40. Describe 3 advantages of passwords. Describe 3 disadvantages.

41. What is a password hash salt? A pepper?

42. What is iterated hashing? What is it used for? How does it compare with other techniques to solve the same problem?

43. Explain the following:

a) Dictionary attack

b) Mangling rules

44. Describe the trade-off that occurs when using system-assigned passwords. How do these passwords help to mitigate dictionary attacks

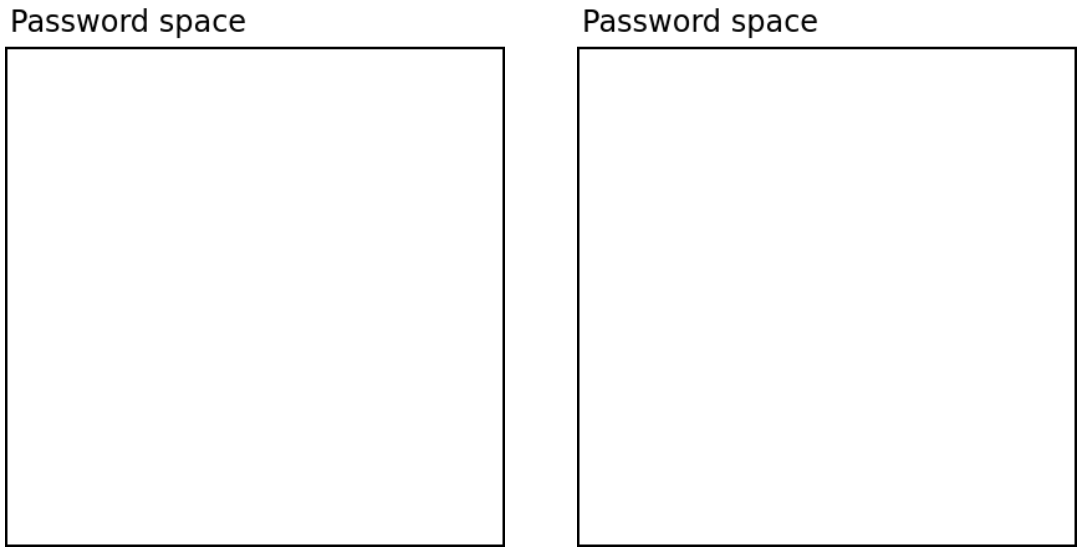
45. Suppose you had a password scheme that has an alphabet of size b and allows passwords as long as n characters. How long would it take to brute force passwords in this scheme in:

a) The worst case?

b) The average case?

46. Consider $q = GT/R$. What does this equation describe? What is each variable for?

47. Draw password distributions in Figure 3.1 according to the captions.



(a) Password distribution we want.

(b) Password distribution we see in practice.

Figure 3.1

48. Discuss rate limiting and password change policies with respect to online guessing attacks. How does $q = GT/R$ factor into making decisions regarding these policies?

49. Discuss the drawbacks of complex site login password composition policies. Suggest at least three better alternatives.

50. What is a passkey? Why are complex passwords preferred for passkeys? How can passphrases help with usability issues associated with these complex passwords?

51. How can password blacklisting be used to reduce the effectiveness of dictionary attacks?

52. Why are secret questions generally a terrible method for password recovery? Why do you think they are so widely used despite their drawbacks?

53. What is a one-time password? How can a Lamport Hash Chain be used to extend a single key word to t one-time passwords?

54. Explain how Lamport Hash Chains are vulnerable to a man-in-the-middle attack using small $n = t - i$.

55. Describe the following categories of authentication and provide an example for each:

a) What you have

b) What you are

c) What you know

d) Where you are

56. What is multi-factor authentication?

57. Describe some advantages and disadvantages of hardware token authentication.

58. Give an example of each of the following modalities with respect to biometric authentication:

a) Physical

b) Behavioral

c) Mixed

59. Biometrics are secrets.

a) True

b) False

60. Biometric authentication to a remote site via a phone sends your biometric signature for authentication.

a) True

b) False

61. Is it better to have a higher false acceptance rate or false rejection rate? Make an argument using the definitions of each.

62. Draw a bimodal distribution with a higher false acceptance rate than false rejection rate. Clearly define where your t is located with a straight line.

63. What is EER? How is it used in practice?

64. What is the difference between identification and authentication?

65. Describe at least three criteria used for the evaluation of biometric authentication.

66. Explain failure to enroll and failure to capture in the context of biometric authentication.

4 Authentication Protocols and Key Establishment

5 Operating Systems Security and Access Control

6 Software Security – Privilege and Escalation

7 Malicious Software

8 Public Key Certificate Management and Use Cases

67. What is a distinguished name? What does it do?

68. What is a public key certificate? What does it do? What trusted third party does it rely on?

69. Which of the following is generally preferred?

    a) An entity sends a public key in their certification request

    b) An entity sends a certification request and the CA generates their key pair

    c) These are equivalent

70. What does a PKI do? Give one example of a PKI and how it is used in practice.

71. What is X509 used for? What does it specify?

72. What is a certificate chain? Why are they necessary?

73. What part of a certificate chain requires implicit trust? What part(s) is expected to sign its own certificate?

74. What happens if part of a certificate chain is broken?

75. What is an out-of-band channel? What do we use them for when checking certificate integrity?

76. Why is allowing users to manually trust certificates not always the best idea? Why do browsers do it anyway?

77. Consider Trust-On-First-Use under the following circumstances. What is the result in each case?

a) An attacker has replaced a self-signed certificate with their own

b) The self-signed certificate is legitimate

78. Suppose you tried to browse a site with an X509v3 certificate that had 4 extension fields, one marked critical, the other three non-critical. What would happen in the following scenarios (assuming the rest of the certificate is fine)?

a) You have an older browser that cannot handle the critical extensions

b) You have an older browser that cannot handle one of the non-critical extensions

c) You have an older browser that cannot handle any of the non-critical extensions

79. Draw a picture of a cross certificate between two hierarchies. Describe one use case for cross certificates.

80. Consider the difference between *trust* and *trustworthiness*. What does a certificate trust model provide with respect to these terms?

81. Label (a), (b), and (c) in Figure 8.1 with the correct label from the following choices:

a) Ring mesh

b) Bridge CA model

c) Separate domains

82. Circle the trust anchor(s) for the right-most leaves in each diagram in Figure 8.1.

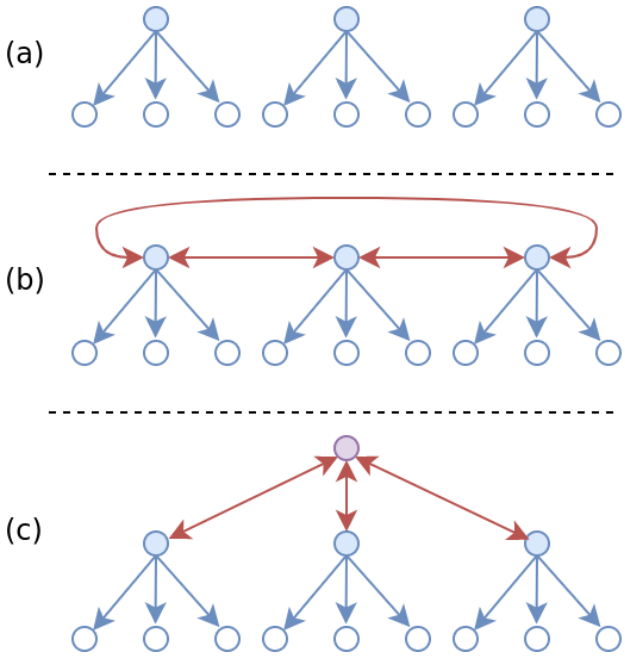


Figure 8.1

83. What are the leaf nodes in the browser trust model for CAs?

84. For each label in Figure 8.2, decide which of the following terms suits it best.

    a) Browser trust model

    b) Strict CA hierarchy model

    c) Enterprise PKI model

85. Circle the trust anchor(s) for the right-most leaves in each diagram in Figure 8.2.

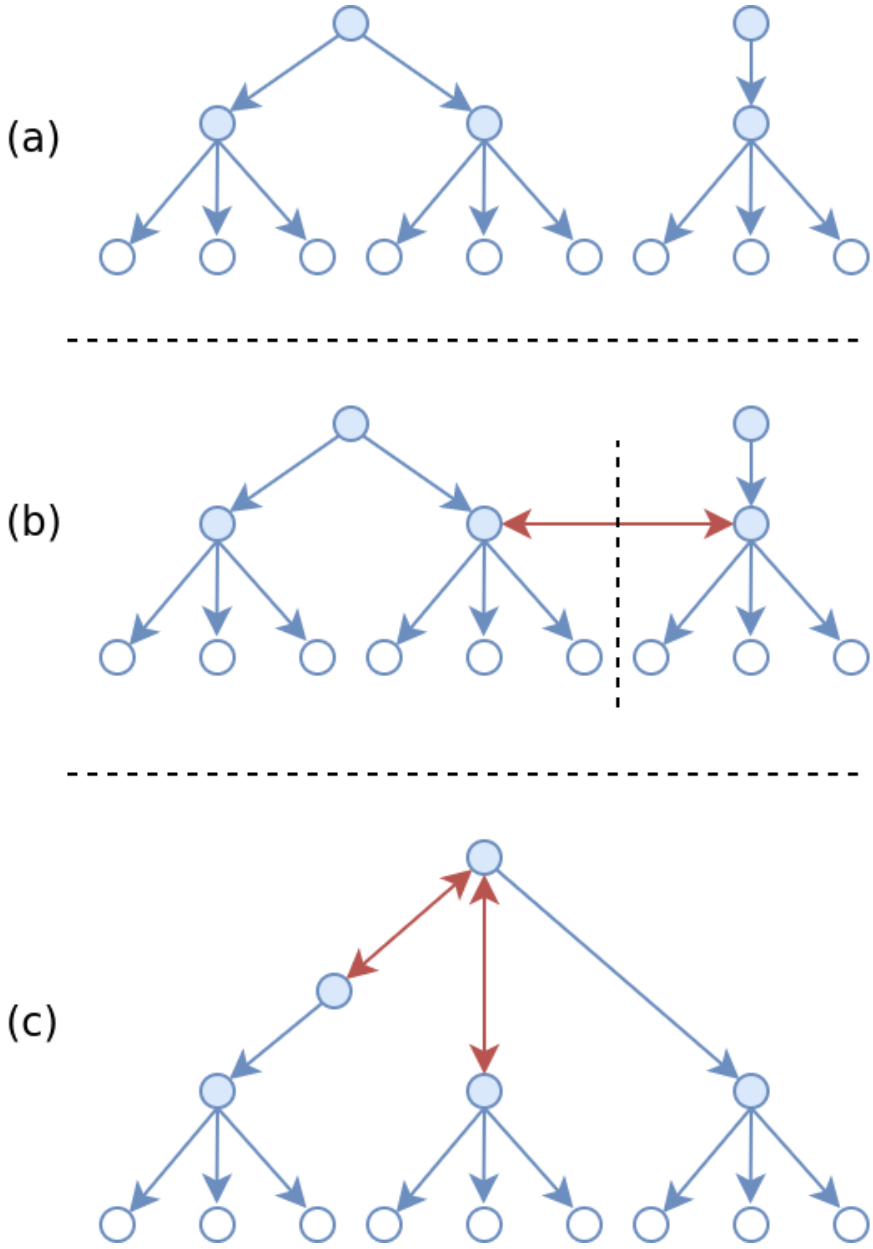


Figure 8.2

86. What is Web of Trust? Give an example of where we might use it.

87. Order the following validation types from least to most thorough:

a) DV

b) OV

c) EV

d) IV

88. What is a certificate substitution attack? What does this mean for the browser trust model?

89. How easy is it for a user to alter the browser trust model? Answer in terms of both accepting and revoking trust. Is this a good thing or a bad thing?

90. What do PEM, PGP, and S/MIME all have in common?

91. Where is the security header added in secure email?

92. Briefly explain how secure email uses public-key and/or symmetric-key cryptography.

93. What is a public key server? What is a certificate directory? What problem do these hope to solve?

9 Web and Browser Security

10 Firewalls and Tunnels

11 Intrusion Detection and Network-Based Attacks

Part II

Notes

12 Basic Concepts and Principles

13 Cryptographic Building Blocks

14 User Authentication

15 Authentication Protocols and Key Establishment

16 Operating Systems Security and Access Control

16.1 Memory Protection

- physical memory -> pages
- virtual memory -> joins pages from all over the place to appear as if they are continuous

Descriptor Register.

- OS maintains per-process descriptor segment that holds a table of indices (segment descriptors)
- descriptor base register points to this segment
- each index has starting address, current size, and access permissions

Memory Access Permissions.

- `r` readable by non-supervisor
- `w` writable by non-supervisor
- `x` executable by non-supervisor
- `m` supervisor mode when executing
- `f` fault bit: trap to supervisor on access (overrides other bits)

16.2 Reference Monitor, Access Matrix, Security Kernel

- rows = subjects
  - each row is a capability list for a subject
- columns = objects
  - each column is an access control list for an object
- reference monitor controls access attempts to resources
  - in practice, this will be several monitors for several access types
- these ideas taken together form a security kernel

- reference monitor is only a model, implementation varies in practice

16.3 Audit Logs

- OS maintains audit logs that describe user attempts to access resources, run programs, etc.
- basically leaves an audit trail of how the user interacts with the system
- these logs can be useful for host-based IDS data collection

16.4 File / Directory Permission Bits

- binary
  - `d rwx rwx rwx`
  - read write execute for user, group, other
  - first bit specifies special file types (directory, symlink, char device, block device, named pipe)
- octal
  - 777 -> `rwx` for all
  - 7777 -> `rwx` for all and also sets setuid, setgid, and sticky bit
  - the left most octal digit specifies: 1 -> sticky bit, 2 -> setgid, 4 -> setuid

`setuid`, `setgid`.

- special bits that specify we run with the euid and egid of the owner respectively
- often used with root-owned binaries
- denoted with an `s` in the character permission model on top of the execute bit for user and group
  - if we don't have execute permissions, we still can't run it though

Real, Effected, Saved `uid` and `gid`.

- `ruid`, `rgid`
  - the actual uid and gid of the user
- `euid`, `egid`
  - effective uid and gid of the user, changed when running a setuid or setgid executable
- `suid`, `sgid`
  - saved uid and gid, used to restore privilege back to normal when dropping privileges

Sticky Bit (`t-bit`).

- prevents the deletion and renaming of files within a directory
  - unless they are owned by you
  - or you are the owner of the directory

`umask`.

- permissions to take away from default permissions on file creation

- for example, if we created a file with
  - rw -> `666` and had a umask `022` (very common)
  - we would end up with `644`
  - this corresponds to rw for owner and r for group and other
- for octal, this is easy, we just subtract them
- for binary, just and it with the notted umask

ACLs.

- augments user group other model
- specify permissions for specific users and groups in addition to `ugo`
- mask specifies limits for these specific users and groups
  - for example, if I set `user:william:rwx` but mask was `mask::r-x`, `william` would not be able to write

16.5 Symbolic Links, Hard Links, Deleting Files

- symbolic link
  - points to pathname of another file
  - deleting original breaks link
- hard link
  - associates new path name with same inode
  - deleting original preserves link
- deleting a file
  - removes a pathname-inode association
  - file is only "deleted" once last link is removed
  - but even then, data still exists, can be recovered
  - only way to truly delete is to overwrite with `0`s then unlink

16.6 Role-Based Access Control / Mandatory Access Control

17 Software Security – Privilege and Escalation

17.1 Race Conditions

- TOCTOU
  - time of check to time of use
  - setuid has owner's privileges, but we need to make sure real user can access the file
  - `access` system call traditionally used
  - but an attacker can attempt to change the file between checking access and opening it for writing
  - this allows us to access another file that we shouldn't
- how to stop this?
  - atomic privilege checks would be nice
    - not portable

    - drop privileges and then fork before opening the file, make descriptor available to parent before exiting
        - also not portable
    - use system calls that deal directly with file descriptors
        - also not portable
    - no perfect solution, application dependent
- other notes
    - niceness can tip the balance in the attacker's favor
    - make checking process much higher niceness (slower) -> larger window to win the race

17.2 Integer-Based Vulnerabilities and C Issues

- problems:
    - signedness
    - overflow
    - underflow

17.2.1 Problems with C

- favor efficiency/direct access over security
    - it lets us make mistakes like the ones above without warning
- undefined behavior -> machine-dependent results
    - signed integers have undefined behavior, unsigned integers use modular wrapping
- pointer arithmetic
- accessing memory using overflow/underflow values
- using overflow/underflow values for branching conditions
- pointer arithmetic changes based on size of type
- types can have different sizes based on architecture

Why We Can't Just Fix This.

- some developers rely on the behavior we described above (intentional overflows)
    - introducing runtime errors breaks backwards compatibility
- warning about possible errors is not necessarily possible
- safe integer libraries are a good option, but how do we enforce adoption?

17.3 Stack-Based Buffer Overflows

- too many bytes written to stack may overflow into adjacent memory
- natural vs intentional
    - natural yields unexpected outcomes
    - intentional results in security issues

17.3.1 Memory Layout

- text and global data segments are lowest, initialized at start
- heap is low, grows up
- stack is high, grows down

17.3.2 How it Works

- on function call, new args are pushed onto stack
- current instruction pointer is pushed as return address
- now we push local vars onto stack and execute the function
- overflowing a local var overwrites higher memory
  - if we overwrite return address, we can point it elsewhere, including into our overwritten memory

17.4 Heap-Based Buffer Overflows

- dynamic allocation is less predictable, varies between systems
- attacker has to experiment to find a vulnerable variable
  - there needs to be a useful higher memory address that is exploitable
- this is also only useful if memory corruption does not crash the program

17.4.1 Heap Spraying

- inject code many, many times into various locations in the heap
- then use an independent exploit to trigger code execution
- you only need to get it right once

17.5 Three Main Steps for Buffer Overflows

1. Code injection
   - inject code into an area of memory
2. Corruption of control flow
   - change control flow to go to that code
3. Seizure of control
   - run that code

17.6 Buffer Overflow Exploit Defenses

- make head and stack non-executable
- stack canaries
  - key words that detect code injection
  - if the key word is overwritten, raise an error
- run-time bounds checking
- ASLR, random heap allocators
  - randomize memory layout

    - disrupts attacks that rely on known memory locations
- type-safe languages
- safe C libraries
- static analysis tools
    - analyze code and flag potential problems

17.7 Privilege Escalation

- move from fixed functionality to a command shell
- escape a sandbox
- gain root privileges
- move from root privileges to kernel privileges (e.g., install a rootkit kernel module)

18 Malicious Software

18.1 Viruses vs Worms

- both propagate, carry a payload
    - mechanisms differ
- virus
    - requires user interaction to propagate, run
    - payload triggers on a condition
    - infect host files / software
        - even documents -> macro viruses
- worm
    - propagate automatically, over networks
    - don't need to be run by a user
    - exploit running software

Virus Infection Strategies.

1. Write beginning of file
2. Write end of file
3. Overwrite entire file
4. Overwrite a specific portion of file

- appending or prepending does not damage functionality, but is noticeable due to file size change
- overwriting harms functionality of program
    - make sure it's not critical to running the system

Virus Detection.

- impossible to write a program to detect all possible viruses
- instead, we play a cat and mouse game with attackers
    - increasing complexity

- detection: data signature-based detection, integrity checking, behavioral signatures

Virus Anti-Detection.

- encrypt and store internal decryptor key
- polymorphic virus that changes decryptor portions across infections
- place decryption key in an external file
- metamorphic virus -> mutates both body and mutation engine

Auto Rooters.

- scan for vulnerable targets, exploit opportunistically
- only need one vulnerability to spawn a root shell and/or install a rootkit

Fast Worm Spreading.

- generate hit lits by scanning for vulnerable hosts
- permute compromised addresses to find new targets strategically
- internet-scale hit lists
  - scan servers

18.2 Trojan Horses

- disguises malicious functionality as useful functionality
- sometimes combines malicious functionality with useful functionality
- often social engineering
  - disguised updates for legitimate programs
- may remain undetected for some time

18.3 Backdoors

- bypass normal entrypoints (and access control)
- remote-access Trojan (spawn a remote root shell)
- malicious remote desktop/shell tools
- rootkits

18.4 Key Loggers

- can be implemented with rootkits hooking system calls
- log user actions / keystrokes to steal information
- more severe -> surveillance software

18.5 Rootkits

- installed after an attack to allow attacker to keep coming back
- goals

    - conceals its presence
    - controls or manipulates applications/OS
    - facilitates long-term additional malicious activity
- rootkits often provide backdoors
- can be implemented in userspace or kernelspace
    - kernelspace is more dangerous (supervisor mode, control whole OS, easier to conceal itself)

Rootkit Methods.

- hooking system calls (patch dispatch table)
- inline hooking
    - detour and trampoline functions
- kernel object modification (to hide activity)
    - either directly change the kernel object
    - or modify the return values of key system calls via postprocessing
- installation strategies
    - exploit buffer overflows in kernel
    - alter shared library or application in userspace
    - install loadable kernel module (after gaining root privilege via some other means)

18.6 Drive-By Downloads

- exploit browser vulnerability + code injection to legitimate site (or by phishing with a malicious site)
- results in downloading a malicious executable silently
- defenses?
    - downloader graphs
    - comparing download patterns

18.7 Droppers

- malicious software whose job it is to download and install other malicious software
- can arrive by means of a worm, virus, trojan, or drive-by-download

18.8 Ransomware

- malware designed to extort users into paying
- often controlled via a botnet
- anonymous payment methods like bitcoin are exacerbating their effectiveness
- examples
    - encryption file lockers (use encryption to lock users out of their files and require payment to unlock the key)
    - other methods of file locking (access control, threaten to erase data, syskeying in Windows)

18.9 Botnets

- a coordinated network of enslaved machines
  - exploited initially via shell code or other means
- attack establishes control over many remote systems
  - the controlling machine is called the botnet header
- uses these systems to commit cyber crimes
  - provide an economy of scale to attackers
  - DDOS attacks, spread ransomware, malicious crypto mining, etc.

18.10 Zero-Day Exploits

- target a vulnerability in a program that was not known previously
- only useful before the vulnerability is patched
- users not patching their software can make this especially problematic

18.11 Logic Bomb

- malicious payload executed when a condition is met
- this is used in viruses and worms for example

19 Public Key Certificate Management and Use Cases

19.1 PKIs

- public key infrastructure
- basic function is to provide a means of verifying a public key belongs to an entity
- CAs, sPKI, WoT

19.2 Certificates

- fields
  - subject (DN)
  - issuer (DN)
  - version information
  - validity period
  - extension fields
    - critical and non-critical (if you can't handle critical, reject)
- X509 tells you what fields
- DN
  - distinguished name (uniquely IDs an actor)
  - Country
  - Organization
  - Organizational Unit
  - Common name

- subject alternate name(s)
  - other names that a subject can go by
- certificate can be
  - issued
  - revoked
  - expired
  - valid
  - invalid

19.3 Certificate Authorities

- issue certificates
- root CA requires implicit trust
  - intermediate CAs to distribute the task of issuing/verifying
- each CA has its own certificate
  - root CA (trust anchor) signs its own

Certificate Chains.

- intermediate CAs
- root of the chain is the root CA
  - root CA is generally trusted via information obtained from an out of band channel
  - e.g., browsers have a list of root CAs they trust implicitly
    - cross certificate pairs can handle the test

Cross Certificate Pair.

- two CAs sign each other's certificates
- if you trust one, you can trust the other one
- builds connected trust on the internet

19.4 Certificate Trust Models

Single-CA.

- each tree has one CA
- Separate Domains
  - one CA for each set of leaf nodes
  - everything is kept completely separate otherwise
  - no cross certificate pairs
- Ring Mesh
  - like separate domains except all top-level CAs are linked with cross certificate pairs
- Bridge CA Model

    - cross certificate pairs with a central hub

Strict Hierarchy.

- root CA with intermediate CAs forming a connected tree
- cross certification is possible within the tree but not between separate trees

Ring-Mesh of Tree Roots.

- like strict hierarchy except now tree roots are free to cross-certify

Browser Trust Model.

- list of trust anchors for each end point
- no cross certificates in browser trust model
- each tree is a hierarchy
- leaf nodes correspond to servers

Enterprise PKI.

- intermediate CAs at lower levels cross certify between departments
- allows fine-grained control of trust

Web of Trust.

- each entity signs their own keys and then entities sign each other's keys
- you pick a subset to trust, and then trust keys trusted by those you already trust
- used in PGP, secure email for example

19.5 Grades of Certificate

- IV
    - user decides to trust a self-signed certificate
- DV
    - validates a domain only
- OV
    - validates domain and organization
- EV
    - extensive validation process

19.6 Secure Email / Public Key Distribution

- technologies
    - PGP -> web of trust
    - PEM -> one-root PKI
    - S/MIME -> secure MIME, used with centralized certificate management
- relies on either key distribution or certificate directories

Security Header.

- goes into the content rather than the original header
  - for backwards compatibility
- this header is digitally signed and often contains an encrypted symmetric key
  - this symmetric key is encrypted with public key crypto
  - allows us to read encrypted mail offline
- steps
  1. verify the signature
  2. decrypt the symmetric key
  3. use the symmetric key to decrypt the body to plaintext

Key Distribution.

- we ask a trusted third party to generate a public key and owner ID for us

Certificate Directory.

- we ask a trusted third party to distribute certificates to use as needed

20 Web and Browser Security

N.B., I am omitting all the webdev review stuff at the beginning of this chapter. If somebody would like to make a PR to add it, that would surely be appreciated.

20.1 TLS and HTTPS

TLS Layers.

- two layers

1. handshake layer
   - (unencrypted) key exchange
   - (encrypted) server parameters
   - (encrypted) integrity and authentication
2. record layer
   - protects all data using negotiated parameters

TLS Key Exchange.

- three options

1. DHE (ephemeral diffie-hellman)
   - either finite fields (integers mod p)
   - or elliptic curves
2. PSK (pre-shared key)
   - either pre-shared via out-of-band channel or from a previous TLS session

3. PSK combined with DHE

TLS Server Authentication.

- two (or technically four) options

1. either PSK session key
2. or digital signature (based on certificate)
   - digital signature RSA
   - or ECDSA
   - or EdDSA

- client and server send MAC codes to signal they are done

TLS Encryption and Integrity.

- usually encryption is done in two ways

1. either via ChaCha20 (stream cipher)
2. or AES (block cipher) using an acceptable mode

- hash algorithms provide data integrity

20.2 DOM Objects

- `location` -> sent by server in HTTP header
- `domain` -> origin server can increase cookie's scope
- `document.domain` -> hostname that document was loaded from
- `window.location.href` -> URL of requested document; modify to redirect

20.3 HTTP Cookies

- HTTP is stateless
- cookies allow persistent state
  - by default, persist over a session (in browser memory)
- multiple cookies can be set by an origin server using headers
  - these cookies are returned from the origin server on later visits
- cookie attributes
  1. max-age/expires -> max age in seconds or expiration data of the cookie
  2. domain -> set scope of cookie to a higher level domain but not TLD
  3. path -> which origin server pages a cookie is returned to
  4. secure -> if specified, use HTTPS (over TLS) instead of HTTP
  5. httponly -> only HTTP can access, no javascript / DOM API access

20.4 Same-Origin Policy

- isolates documents

- document from one origin should not be able to interfere with or manipulate a document from another origin
- prevent javascript from being used by a malicious page to redirect to a sensitive page and access or modify data
  - recall from the assignment

SOP Rules.

1. assign an origin to a document based on URL
2. scripts and images are assigned an origin based on who loaded them (not who they are retrieved from)
3. scripts may only access content whose origin is the same as their own

Problems.

- goals are hard to capture precisely
- difficult to enforce

How to Identify an Origin.

- based on a 3-tuple of information
  1. scheme
  2. host
  3. port
- content from distinct origins needs to be in new frames or windows
  - `<iframe src="url">`
  - `window.open("url")`

Relaxing SOP.

- developers can change `document.domain` to classify domains sharing the same suffix as belonging to the same origin
- this is bad from a security perspective

How Things Are Different For Cookies.

- cookies use different attributes for origin identification
  - they no longer use port
  - they use `secure` and `httponly` in place of `scheme`

20.5 Authentication Cookies

- server stores session ID in a cookie
- user logs in once, and cookie authenticates them next time they visit the site for the duration of the session
- server may specify a session expiration time
- persistent cookies may extend authentication beyond the lifespan of the browser window

Cookie Theft.

- javascript, proxies, client-side malware, physical/manual filesystem access
  - steal cookie, use to authenticate as the victim
- session hijacking (not the same thing as TCP session hijacking)
- defense against javascript is to set `httponly` in the cookie
  - this does not work against the other attacks

Cookie Protection.

- the server should encrypt and either MAC or sign the cookie to provide data integrity and data origin authentication
- server will then decrypt and verify the cookie when it comes back

20.6 CSRF

- cross site request forgery

20.7 XSS

- cross site scripting
- insert HTML script tags that cause malicious javascript to be executed

XSS-Stored.

- when the page is rendered based on content stored on the server
- e.g. a comments section
- loaded persistently whenever a user accesses the site

XSS-Reflected.

- exploit parameterized script to insert XSS
- e.g. a file not found error message that pops up based on requested path
  - change requested path to a script tag

Possible Impacts.

- cookie theft
- redirection to malicious site
- altering the contents of the site
- seizure of browser control

20.8 SQL Injection

- common tricks
  - terminate quotes early by adding a quote of your own
  - terminate a statement by adding a `;` (works well after integers)

  - insert a comment with `--` to eliminate query checks
  - insert `OR` that always resolves to true followed by `--`

20.9 Sanitization

- used to defeat XSS, SQL injection
  - this is a lot harder than just sanitizing `<script>`, `;`, `"`, etc.
  - there are a lot of factors to consider
  - for example, evasive encoding is a problem that may not seem obvious at first

For SQL Specifically.

- adjust input to remove bad characters
- blacklisting known-bad input
- only allowing known-good input (whitelisting)

21 Firewalls and Tunnels

- inbound (ingress) packets
  - protect network from internet
- outbound (egress) packets
  - protect from compromised machines or malicious insiders (accessing the outside world)

As a Chokepoint.

- requires secure perimeter
- this isn't really a thing anymore thanks to mobile data, etc
- but firewalls remain useful
  - protect legacy apps
  - enforce IP security policies lacking wireless access to physical hosts

Limitations.

- assume true perimeters exist
- insider the network is trusted, little protection from users cooperating with outsiders
- trusted users make bad connections
- tunneling can bypass them
- can't inspect encrypted content

21.1 Packet Filter Firewalls

- actions
  - `ALLOW` -> allow packet
  - `DROP` -> silently drop (type-1 deny)

    - `REJECT` -> drop and notify (type-2 deny)
- rules are based on 5 TCP/IP Header fields
    - src address
    - dest address
    - src port
    - dest port
    - protocol (TCP/UDP/ICMP)

Stateless vs Stateful.

- stateless
    - packets processed independently of others
- stateful
    - remembers previous packets when processing
    - state is kept in a state table
- dynamic
    - rules change on the fly

21.2 Proxy Firewalls and Firewall Architectures

- two properties desired
    - transparency
    - performance

Circuit-Level.

- checks TCP 3-way handshake, not packets themselves
    - syn
    - syn-ack
    - ack
- most common implementation
    - client-side library and proxy server daemon `sockd`
    - client-daemon network protocol called `SOCKS`

Application-Level.

- firewall establishes a connection on your behalf and inspects incoming data packet
- can block packets entirely or even alter packet payloads
- requires detailed knowledge of specific application-level protocols

Internal Firewalls.

- gateways between internal subnets
- bastion hosts
    - defensive host exposed to a hostile network
- dual-homed host

  - two network interfaces
  - one address per interface
  - can be suitable for a circuit-level proxy

Enterprise Firewall Architectures.

- screening router (router with packet filtering) + a bastion host
- another strategy uses a perimeter network (network DMZ), a subnet between internal and external network
- another possibility: bastion host in the DMZ between two screening routers (one internal, one external)

21.3 SSH

- secure shell
  - does not provide a shell itself
  - provides an encrypted tunnel to *get to a shell* (or another application)
- before SSH, we sent cleartext over the network -> this is *bad*

Three Protocols -> Three Layers.

- transport layer protocol
  - encrypts and provides integrity
  - includes negotiation of crypto parameters and keys
- user authentication protocol
  - handles SSH authentication, runs over transport layer protocol
- connection protocol
  - single SSH connection for multiple purposes
  - each instance gets a logical channel
  - channels support
    - interactive sessions
    - connection forwarding

Client Authentication.

- during transport layer negotiation, server decides which authentication methods it will accept
  - password
  - kerberos ticket
  - client public key
- for public key login
  - server receives client public key and signature
    - verifies the signature relative to session data (incl. ID)
    - checks that public key is in list of allowed public keys

Server Authentication.

- in non-enterprise, usually it's TOFU
  - if the key we choose to trust is okay, we're all good
  - otherwise, we have connected to a middle person
  - generally this gamble is acceptable if we have no backing PKI
- to make TOFU better, we can cross-check the host's fingerprint hash against one obtained in an out of band channel
- PKI support is the best, two options
  - client database of SSH keys
  - CA-certified SSH keys

SSH Tunnel.

- `-L localport:host:hostport remotehost`
  - listen on local machine at port `localport`
  - forward it to `host:hostport` from `remotehost`
- `-R` same thing but backwards (remote to local instead of local to remote)

21.4 VPNs and Encrypted Tunnels

Tunneling.

- data stream on one protocol travels inside another
- encapsulation
- not necessarily a higher level protocol, encapsulating a lower level
- two technologies often used
  - ssh -> lightweight
  - IPsec -> heavier

VPN.

- physical private network
  - accessed only by trusted users
  - physical isolation, access control, firewalls/gateways
- VPN is a private network created via encrypted tunnels and special protocols

Site-to-Site VPNs.

- bridge private networks across a public channel

Remote Access VPNs.

- authorized clients remote access to a private network
- experience is transparent to physically located

22 Intrusion Detection and Network-Based Attacks

22.1 Response Types: Detection vs Prevention

- Intrusion Detection System
  - detect intrusions and issue an alarm
  - for example, to a sysadmin
- Intrusion Prevention System
  - take active steps to prevent the attack
  - this is not always possible, depends on design, locality, and other limitations
  - e.g., external sensors in a NIDS may not be able to respond in time since they have to copy traffic

22.2 Architecture: Host-Based vs Network-Based

Host Based.

- collect event data on a local machine
- can be used cooperatively across multiple machines on a network
- checking network events on a local machine is still host-based, not network-based

Network Based.

- collect events on a network
- for example, on a specialized gateway device

22.3 Event Outcomes

False Positives.

- when an IDS mistakenly flags a valid event as invalid

False Negatives.

- when an IDS mistakenly flags an invalid event as valid

True Positives.

- when an IDS correctly identifies an invalid event

True Negatives.

- when an IDS correctly identifies a valid event

Base Rates and FPR/FNR Trade-Off.

- base rates

    - if the base rate of incidents is low, very low FPR might produce a lot of false positive in practice
    - people tend to ignore this when thinking about FPR and FNR -> it's a logical fallacy
    - base rate fallacy -> ignore base rate when calculating conditional probabilities
- trade-off between FPR and FNR
    - hard to get both low FPR and FNR
    - IDS systems generally have to make compromised based on use case and desired level of cautiousness

22.4 Methodologies: Anomalies vs Signatures vs Specifications

Blacklisting vs Whitelisting.

- blacklisting
    - specifying disallowed behavior and flagging matches
- whitelisting
    - specifying allowed behavior and flagging mismatches

Anomaly Detection.

- learned profiles used to generate white lists
- alarm when an event deviates from normal

Signatures.

- expert-defined blacklist
- alarm when behavior matches blacklist

Specifications.

- expert-defined whitelist
- alarm when behavior does not match whitelist

22.5 Sniffers, Reconnaissance Scanners, Vulnerability Scanners

- these all have both white-hat and black-hat uses

Sniffers.

- tools need to process packets at line speed (to avoid slowdowns or self-inflicted DoS)
- monitor network traffic (even passively)
    - look at things like headers
    - for unencrypted traffic, we can even look at contents
- attackers can use these tools too
    - this is a motivation for encrypted traffic

Hubs vs Switches.

- hubs broadcast to all neighbors
- switches selectively forward

Reconnaissance Scanners.

- common precursor to an attack
- also useful for sysadmins to map their own network and get an idea of what is going on
  - network introspection/visibility
- send probe connection requests to look for open ports and running services
- port can be
  - open (daemon waiting)
  - closed (no service offered)
  - blocked (denied by access control)
- OS fingerprinting
  - port scanners can identify an OS with high confidence

Vulnerability Scanners.

- scan a system for known vulnerabilities
- produces a comprehensive report
- does so using benign payloads (unlike penetration testing)
- limitations/cautions?
  - responsible release of exploit modules -> attackers already have, so legitimate parties might as well
  - credentialed scans can produce far more detailed results (but be careful)

22.6 DoS Attacks

- DoS attacks either:
  - exploit implementation flaws
  - or exhaust resources by overwhelming

Local vs Remote.

- local
  - not necessarily network-based
  - malware the consumes a lot of resources
  - overflow in a kernel function
- remote
  - poison packets (malformed packets that trigger implementation errors)
  - send false source IP addresses to defeat simple anti-flooding mechanisms
  - ICMP flooding (spamming pings -> ICMP echo requests)
  - UDP flooding (bombard random ports with UDP packets, many of these ports may be closed)

DDoS.

- distributed denial of service
  - DoS attack from many machines flooding packets to a network
  - botnets are often used for this

Ingress/Egress Filtering.

- packet filters to mitigate address spoofing
- use a whitelist and only allow packets in a known range to enter
- martian addresses -> invalid source address
  - these are dropped
- egress filtering network traffic as it leaves
  - match only known destinations
  - prevents hosts being used as attack agents

22.7 Address Resolution Attacks

- DNS servers used to query for IP addresses matching host names
  - go up the chain until we get an answer
- potentially caching at each step
  - cache is cleared after specified TTL
- optionally, we also use static mappings in our local "hosts" file

DNS Attacks (Pharming).

- a means of falsifying DNS lookup at any stage in the process
- pharming forces a user to access a malicious site
  - whereas phishing lures a user into clicking a link
- some examples
  - modify local hosts files
  - tamper intermediate DNS servers
  - modify responses from network
  - malicious DNS server settings (point to a malicious DNS)

ARP Spoofing Attacks.

- ARP maps IP addresses to local MAC addresses on a local network
- hosts broadcast queries w.r.t which MAC belongs to which IP
  - each LAN host keeps a cache table of responses
- attacking hosts can send false ARP replies -> poisons the ARP caches
- attacker can inspect and even modify traffic before it reaches the real destination
- defending ARP spoofing?
  - maintain static read-only ARP tables at each host
  - preferred long term solution -> reliable authentication between local hosts