Computer Security

2019 Exam 2 Review

Paul Krzyzanowski

Rutgers University

Spring 2019

Question 1 version B: 2, C: 3

What is an example of a restriction that *application sandboxes* can provide but namespaces cannot?

(a) Disallow access to the network.
(b) Access only a specific subtree of the file system.
(c) Open only files with a `.docx` suffix.
(d) Isolate the process so it does not see other process IDs.

(a) Namespaces can give a process its own network address … or none

(b) Namespaces have *chroot* functionality

(c) Namespaces cannot filter filenames

(d) Namespaces allow private process ID spaces.

Question 2 version B: 3, C: 4

Application sandboxing provides this advantage over DAC and MAC mechanisms:

(a) It allows specific applications to override the restrictions of MAC/DAC permissions.
(b) It enables access permissions to be set on a per-user basis: different users can have different permissions.
(c) It allows the user, rather than an administrator, to manage access permissions.
(d) It enables per-application access restrictions in addition to the existing MAC/DAC mechanisms.

(a) Sandboxes can restrict operations but not override restrictions that are in place.

(b) Sandboxing is per app, not per user.

(c) DAC are user-managed too.

(d) Yes – additional per-app access restrictions can be defined.

Question 3 version B: 4, C: 5

Seccomp-BPF has this functional advantage over system call interposition (as used in Janus, for example):

(a) It allows validation to take place by a separate user process that can be customized as needed.
(b) It can filter network operations in addition to file operations.
(c) It can examine the content of files.
(d) It is not vulnerable to TOCTTOU attacks.

(a) Janus does this.

(b) Most sandboxing solutions enable control of various system calls, including network-related ones

(c) Scanning content would be costly – none do it.

(d) Janus needs to mimic the state of the OS based on system calls and any changes to the environment.

Question 4 version B: 5, C: 1

Which sandbox analyzes the code prior to execution?
(a) Chromium Native Client (NaCl).
(b) Janus.
(c) Linux seccomp-BPF.
(d) Apple Sandbox.

NaCl scans the code before loading to ensure that jump addresses are aligned to proper boundaries, only allowable instructions are used, etc.

Question 5 version B: 1, C: 2

What component of the Java sandbox keeps a malicious class from replacing methods in standard classes?

(a) Class loader.
(b) Bytecode verifier.
(c) Security manager.
(d) Java virtual machine.

(a) The class loader validates that classes do not overwrite existing standard classes

(b) The bytecode verifier checks the Java bytecode to ensure instructions are valid, there's no self-modifying code, no pointer dereferencing, etc.

(c) The security manager enforces resource access rules at various methods (e.g., java.io, java.net).

(d) Makes no sense

Question 6

version B: 7, C: 8

A *spear phishing* attack differs from a phishing attack because it
(a) Tricks victims into sharing sensitive information.
(b) Is personalized to the victim.
(c) Appears to come from a trusted source.
(d) Contains a malicious attachment.

(a) A phishing attack does this.

(b) Spear phishing is targeted to individual users or groups.

(c) Phishing attacks try to pretend to come from trusted sources too.

(d) It might, but so might a phishing attack.

Question 7

version B: 8, C: 9

In his seminal paper, *Reflections on Trusting Trust*, Ken Thompson suggests that:

(a) You should trust a program only after examining its source code to ensure it contains no malware.

(b) Source code examination is not sufficient; you should check the source of the compiler as well.

(c) Compiler source examination is insufficient since malware can be introduced during compilation.

(d) No source code examination is sufficient since malware can always be injected at runtime.

(c) The compiler can be compiled to inject malware generation when the compiler itself is compiled.

Question 8

version B: 9, C: 10

Trojans differ from backdoors because:
(a) They are willingly installed by users.
(b) They don't need to bypass authentication mechanisms.
(c) They are designed to propagate to other systems.
(d) They reside at the kernel level instead of in a user program.

(a) Apps with backdoors are generally willingly installed too. They're both covert mechanisms.

(b) Backdoors, by definition, bypass authentication mechanisms.

(c) Neither are.

(d) Either of these might – but neither has to.

Question 9 version B: 10, C: 11

Rootkits are:

(a) Software that enables malware to stay hidden.

(b) Malware that runs with administrative (root) privileges.

(c) Malware that runs within the kernel.

(d) Libraries that enable malware to elevate its privilege level.

Rootkits hide the presence of software

– Command modification, shared library changes, or kernel changes

– Hide certain files and/or processes from inspection

Question 10 version B: 11, C: 6

An advantage of behavior-based malware detection over signature-based techniques is:

(a) It does not bother alerting the user if malware is installed but not doing anything.

(b) It can detect never-before-seen viruses.

(c) It is more reliable at identifying malicious software.

(d) It is faster at locating malware.

(a) That's not an advantage!

(b) Yes – it's trying to detect anomalies rather than match pieces of code

(c) No – it's less reliable since it is often hard to define what constitutes malicious behavior

(d) No – it's a slower process

Question 11 — version B: 6, C: 7

The goal of a *malware packer* is to:

(a) Encrypt user files for ransom.

(b) Enable malware to be installed within another application already on the system.

(c) Modify the malware each time it propagates.

(d) Avoid having anti-virus software detect the malware.

A malware packer is code within malware that extracts the malware prior to execution.

Two functions:

(1) Compress (sometimes) to make it easier to transport.

(2) Obscure via XOR, compression, or encryption so signatures can't be detected as easily.

Question 12 version B: 12, C: 14

Perfect secrecy is rarely attained in practice because:
(a) All cryptographic algorithms have inherent weaknesses.
(b) Computer systems have vulnerabilities that can be exploited.
(c) Effective key management and distribution is difficult.
(d) Humans are likely to leak keys.

Perfect secrecy means that nothing about the message is revealed in the ciphertext. The ciphertext can conceivably decode into any possible plaintext.

Claude Shannon proved that perfect secrecy can only be achieved if the key is as long as the message.

Question 13 version B: 14, C: 15

A *stream cipher*:
(a) Does not use a keystream generator.
(b) Is not a symmetric cipher.
(c) Does not implement an SP network.
(d) Does not approximate the one-time pad.

(a) A keystream generator is a pseudorandom sequence of bytes that provides the key for the cipher.

(b) The same key is used to decrypt the message.

(c) It does not use an SP network – just XOR of the keystream data.

(d) If the keystream was a true random sequence of bytes, it would be a one-time pad.

Question 14 version B: 15, C: 16

In cryptography, *confusion* refers to:

(a) The inability of an attacker to identify what encryption algorithm was used.

(b) The percentage of bits of ciphertext that will change when a bit of plaintext is changed.

(c) The lack of a relationship between bits of the key and bits of the ciphertext.

(d) Ciphertext that can decode to multiple valid plaintext messages.

Confusion = difficulty of finding any relationship between a bit of the key and a bit of the ciphertext.

Diffusion = dispersion of the change of one bit of plaintext among the generated ciphertext.

Question 15 version B: 16, C: 17

Which cipher does NOT have an iterative structure (i.e., multiple rounds)?

(a) RSA.
(b) AES.
(c) Twofish.
(d) DES.

RSA is based on a math function:

$$c = m^e \bmod n$$

AES & DES are iterative and use multiple rounds.

We did not study Twofish (derived from Blowfish) but Twofish, Blowfish, and DES are all Feistel network ciphers.

Use the process of elimination!

Question 16 version B: 17, C: 18

Unlike Cipher Block Chaining (CBC) and Counter (CTR) modes, the *Electronic Code Book* (ECB) mode:

(a) Makes each encrypted block a function of the previous encrypted block.
(b) Adds a message authentication code to ensure integrity.
(c) Uses a lookup table for high-speed encryption.
(d) Encrypts each block independently of its position in the message.

(a) That's CBC.

(b) None of them do that.

(c) This won't work for any modern ciphers: too much data.

(d) Yes – it encrypts each block in isolation.

Question 17

version B: 18, C: 19

Trapdoor functions are useful for:
(a) Message authentication codes.
(b) Hash functions.
(c) Keystream generators.
(d) Digital signatures.

Trapdoor functions have an inverse only if you have extra information (the trapdoor).

(b), (c) do not have a need to be invertible.

(a) only uses symmetric operations (e.g., symmetric crypto)

Digital signatures need to be ceated by one party and verified by another party and use public & private keys.

Question 18 version B: 19, C: 12

Forward secrecy requires:
(a) A unique key per session that is encrypted with a long-term key.
(b) All keys used during the session never to be reused.
(c) A key that is as long as the message being encrypted.
(d) The encrypted message to be dependent on the previous message.

Forward secrecy means that past or future session keys will not be compromised if a long-term key (private key) is compromised.

Hence, we cannot encrypt a session key with a long-term key.

Instead, use throw-away keys like Diffie-Hellman to exchange a random throw-away session key.

Question 19 version B: 12, C: 13

For Bob to send a session key to Alice, he would encrypt it with:

(a) Bob's public key.

(b) Bob's private key.

(c) Alice's public key.

(d) Alice's private key.

Bob needs to encrypt data that only Alice can decrypt.

Only Alice has her private key, so Bob needs to encrypt the message in a way that could only be decrypted with Alice’s private key.

Question 20 version B: 21, C: 22

The *Needham-Schroeder key exchange* algorithm avoids replay attacks by:
(a) Using timestamps.
(b) Using nonces.
(c) Using sequence numbers.
(d) Encrypting all data.

Needham-Schroeder uses encrypted nonces.

Each new session contains a new nonce.

Replay attacks to get past the setup session are possible only if a previous session key has been leaked.

Question 21 version B: 22, C: 23

Which key exchange technique does NOT rely on a trusted third party?
(a) Needham-Schroeder.
(b) Denning-Sacco.
(c) Diffie-Hellman.
(d) Kerberos.

All key exchange algorithms that do NOT rely on trapdoor functions need to go through a trusted third party

- Third party has everyone's keys
- Anyone can encrypt a message with their secret key and the third party can read it

Question 22 version B: 23, C: 24

When Alice receives a ticket from Kerberos, it is encrypted with:
(Alice is trying to talk to Bob)
(a) Bob's secret key.
(b) Alice's secret key.
(c) A shared secret session key.
(d) Kerberos' secret key.

Alice gets two things back from Kerberos:

1. A session key that is encrypted with her secret key
2. A *ticket* – the same session key encrypted with Bob’s secret key

Question 23 version B: 24, C: 25

Unlike collision resistance, *pre-image resistance* means:

(a) You cannot find a new message *M'* where *H(M)=H(M')* for some given *M*.

(b) You cannot find two messages M_1, M_2, where $H(M_1) = H(M_2)$.

(c) You cannot derive M when given its hash, *H(M)*.

(d) M is encrypted before it is hashed: *H(E(M))*.

A *pre-image* refers to the original message.

Pre-image resistance means that we cannot derive the original message when presented with its hash.

(a), (b): collision resistance

Question 24 version B: 25, C: 20

A *message authentication code* (MAC) differs from a hash because:
(a) It is data that is distinct from the message.
(b) It is reversible.
(c) It is a fixed size value.
(d) It requires a key.

(a) A hash contains data that is distinct from the message too

(b) Neither are reversible.

(c) Both are fixed-size.

(d) A MAC is a keyed hash: you can validate the hash only if you know the secret key.

Question 25 version B: 20, C: 21

If you manage to get Alice's digital certificate, you **cannot**:

(a) Validate messages that are digitally signed by Alice.

(b) Impersonate Alice.

(c) Create a message that only Alice can read.

(d) Extract Alice's public key.

(a) You can – using the public key in her certificate.

(b) You cannot – you need Alice’s private key to do that.

(c) You can encrypt a message with Alice’s public key.

(d) The public key is in the certificate.

The end