Wagner
Spring 2014

CS 161
Computer Security

Midterm 2

Problem 1 ***True or False*** **(16 points)**

Circle True or False. Do not justify your answer.

(a) True or [✓ False]: Firewalls are commonly deployed because they never affect functionality.

Solution: False. Firewalls can affect some functionality (e.g., some peer-to-peer software).

(b) True or [✓ False]: No intrusion detection system is capable of detecting novel attacks.

Solution: False. Anomaly detection can potentially detect (some) novel attacks, as can behavioral detection.

(c) True or [✓ False]: Full TCP reassembly is sufficient to implement a network intrusion detection systems (NIDS) that stops all evasion attacks.

Solution: False. There are other kinds of evasion attacks, e.g., at application-layer semantics.

(d) [✓ True] or False: A host-based intrusion detection system (HIDS) is harder to evade than a network intrusion detection system (NIDS) because a HIDS has access to application-layer semantics.

Solution: True. A HIDS has more visibility.

(e) True or [✓ False]: The false positive and false negative rates of a given intrusion detection system provide enough info to classify it as “good” or “bad” at detecting threats.

Solution: False. This is the base rate fallacy. It also depends upon the base rate of attacks (and typically on the cost of a false positive vs of a false negative).

(f) True or [✓ False]: The Kaminsky attack is an on-path attack; off-path attackers cannot mount a Kaminsky attack.

Solution: False. It is an off-path attack. (An on-path attacker wouldn't need any of the cleverness in the Kaminsky attack; they could see the DNS transaction ID and thus could directly spoof replies without any clever tricks needed.)

(g) [✓ True] or False: On-path attackers can successfully eavesdrop on the data sent over a TCP connection.

Solution: True. They can see all the packets in the connection.

(h) True or [✓ False]: Off-path attackers can successfully eavesdrop on the data sent over a TCP connection.

Solution: False. They're not in a position to see the packets.

Problem 2 ***More True or False*** **(16 points)**

Circle True or False. Do not justify your answer.

(a) [✓ True] or False: On-path attackers can successfully tamper with (i.e., modify) the data sent from Alice's web browser to `http://www.cnn.com/`.

Solution: True. They can observe the sequence numbers and port numbers, which is all that is needed to inject false packets or mount a TCP hijacking attack.

(b) True or [✓ False]: Off-path attackers can successfully tamper with (i.e., modify) the data sent from Alice's web browser to `http://www.cnn.com/`.

Solution: False. On a modern TCP stack, they won't be able to predict the sequence numbers, so it won't be feasible to mount TCP hijacking attacks.

(c) True or [✓ False]: DHCP spoofing only affects the integrity of DNS lookups and has no practical effect on web security.

Solution: False. It has a severe effect on the security of all web sites that use HTTP: the same-origin policy is defined in terms of domain names (not IP addresses), so defeating DNS lets a malicious site defeat the same-origin policy.

(d) [✓ True] or False: Diffie-Hellman is secure against passive eavesdroppers who cannot modify packets or send forged packets.

Solution: True. That's what it is designed for.

(e) True or [✓ False]: Diffie-Hellman is secure against man-in-the-middle attacks.

Solution: False. We went over a man-in-the-middle attack in lecture. To defeat man-in-the-middle attacks, you need some kind of authentication (e.g., signed Diffie-Hellman).

(f) [✓ True] or False: Cryptographic hash functions are required to be one-way and collision-resistant.

Solution: True. That's part of the definition of what it means to be a cryptographic hash function.

(g) True or [✓ False]: The IV in CTR mode (counter mode) must be kept secret.

Solution: False. It is sent as part of the ciphertext, so it is visible to any eavesdropper.

(h) Alice and Bob share a symmetric key k. Alice sends Bob a message stating, "I owe you \$100", and also sends a message authentication code (MAC) on this message computed using the key k.

True or [✓ False]: Assuming the MAC algorithm is secure, Bob can now go to his bank and prove to the bank teller that Alice does indeed owe him \$100 by giving his key k to the teller.

Solution: False. Since Bob knows the key k, he could have made up the message and computed the MAC tag on it himself.

Problem 3 *Vulnerability mitigation* (10 points)

Recall that the version of VSFTPd we used in Project 2 had a widely known vulnerability ("smile at login"). You're the sysadmin for a large company, and you've learned that there is an internal server inside your company's network running this version of VSFTPd. Unfortunately, you can't modify that server.

(a) Describe one way you could use a stateful packet filter to prevent an outside attacker from exploiting the VSFTPd vulnerability.

(Hint: FTP uses TCP and is unencrypted.)

Solution: There are many possible answers:

- An excellent answer: Scan all packets, reassemble the TCP stream, and look for `:)` in a username; if you see this, block the rest of the connection.
- Another excellent answer: Block all connections to port 6200 on that server, i.e., to the backdoor port.
- Almost as good: Scan all packets and block any connection if it contains `:)` in a username. (Still a very good answer, but we gave slightly fewer points to this, to give credit to those who noted the need to reassemble the TCP stream first.)
- Also good: Block all inbound FTP connections from any external host to this VSFTPd server. (This answer is not quite as good because it prevents external use of the server, so we awarded most of the points but took off a couple of points for the potential loss of functionality.)

Other answers that impacted functionality more got correspondingly fewer points (e.g., block all access to FTP on all servers; block all access to that VSFTPd server; block all inbound connections to internal servers; block `:)` anywhere in any TCP data stream).

(b) Suppose that we have deployed a stateful packet filter (as in part (a)) at the border of our network, where it connects to the rest of the Internet. Now imagine that an employee's laptop has been infected with malware. Could the malware exploit the VSFTPd vulnerability? Why or why not?

Solution: Yes. The malware can wait until the employee's laptop is on the internal network and then make a connection from the internal laptop to the internal VSFTPd server. This connection never goes through the stateful packet filter, which thus can't block the attack.

Problem 4 ***Password hashing*** **(12 points)**

Joe runs a large website that allows users to log in and share images. When a new user sets up their account, the website hashes their password with SHA256 and stores the hash in a database. When a user logs in, the website hashes the supplied password with SHA256 and compares it to the stored hash.

Joe figures that with this scheme, if anyone hacks into your database they will only see hashes and won't learn your users' passwords. Out of curiosity, Joe does a Google search on several hashes in the database and is alarmed to find that, for a few of them, the Google search results reveal the corresponding password. He comes to you for help.

(a) What mistake did Joe make in how he stored passwords?

Solution: He didn't use a salt.

(His other mistake was to use a hash that is too fast, though that doesn't really explain why the hash turned up in a Google search, so this didn't receive full credit.)

(b) What is the consequence of this mistake? In other words, what is the risk that it introduces and how many of Joe's users could be affected? Does it affect only users whose password hashes are available in Google search, or does it go beyond that?

Solution: If the database is leaked (e.g., server compromise), the attacker can mount offline password guessing attacks. Such an attacker might be able to recover many of the users' passwords—not just those whose password hashes are listed in Google search.

(c) How should Joe store passwords? More specifically, if a user's password is w, what should Joe store in the database record for that user?

Solution: $s, F(w, s)$ where s is a random salt chosen independently for each user and where F is a slow cryptographic hash, e.g., SHA256 iterated many times ($F(x) = H(H(\cdots(x)\cdots))$ where H is SHA256).

Problem 5 ***Deja vu ... all over again*** **(17 points)**

The C code below should look very similar to something you've seen in Project 2. You have the same goal—you want to write an exploit that spawns a shell. For your reference:

shellcode (as seen in gdb): `0x895e1feb, 0xc0310876, 0x89074688, 0x0bb00c46, ...`
ASCII: A-Z: `0x41-0x90`, a-z: `0x61-0x7a`, newline: `0x0a`, escape: `0x1b`, tab: `0x09`

```
1:  void deja_vu()
2:  {
3:    char door[8];
4:    gets(door);
5:  }
6:  int main()
7:  {
8:    deja_vu();
9:    return 0;
10: }
```

(a) In gdb, what line should you set a breakpoint at to check if your exploit succeeded, without any additional `step` calls?

Solution: Line number: 5

(b) The relevant gdb output below is from **before** you feed in your malicious input.

```
(gdb) x/8x door
0xbffff9b8: 0xbffffa7c  0xb7e5f225  0xb7fed270  0x00000000
0xbffff9c8: 0xbffff9d8  0x0804842a  0x08048440  0x00000000
(gdb) i f 0
Stack level 0, frame at 0xbffff9d0:
 eip = 0x8048412 in deja_vu (dejavu.c:7); saved eip 0x804842a
 Locals at 0xbffff9c8, Previous frame's sp is 0xbffff9d0
 Saved registers:
  ebp at 0xbffff9c8, eip at 0xbffff9cc
```

What should you see if you run `x/8x door` **after** the malicious input has been fed in but before the shellcode executes? Assume that the exploit was successful, and the bare minimum was changed in memory. Your solution should have at least one 4-byte chunk of shellcode.

Solution:

```
0xbffff9b8: 0xbffffa7c  0xb7e5f225  0xb7fed270  0x00000000
0xbffff9c8: 0xbffff9d8  0xbffff9d0  0x895e1feb  0xc0310876
```

or

```
0xbffff9b8: 0xbffffa7c  0xb7e5f225  0xb7fed270  0x00000000
0xbffff9c8: 0xbffff9d8  0xbffff9d4  0x08048440  0x895e1feb
```

(c) Recall that in Project 2, your absolute memory addresses were randomized depending on your login. Explain how this kind of randomization could make exploiting the above vulnerability more complicated, if you got an unlucky choice of memory addresses. Provide an example of a specific address for `door` which would cause this problem. Your example address must be between `0xbfff0000` and `0xbffffffc`.

Solution: Address of door: `0xbfff0a08`

Explanation: We'll need to overwrite the return address with an address like `0xbfff0a20`. This address contains a `0x0a` byte, so we'll need the malicious input to contain a `0x0a` byte, followed a bit later by the shellcode. However, `gets()` stops reading the input when it sees a `0x0a` byte, which will mess up our exploit.

There are many other example addresses that would force the buffer to contain a `0x0a` byte.

We also accepted anything that would force the buffer to contain a `0x00` byte.

Problem 6 ***A special encryption algorithm*** **(7 points)**

A security company has determined that, even using a bunch of fast computers, brute-force attacks can break at most a 50-bit key in a reasonable amount of time. So, they decide to design an encryption algorithm that will encrypt 64-bit messages under a 64-bit key. They already have a good block cipher E that can encrypt a 32-bit message under a 32-bit key and that is resilient against all attacks except brute-force attacks. So, for the 64-bit encryption algorithm, they simply split the message M in two parts M_1 and M_2 and the key K in two parts K_1 and K_2. The ciphertext C is then computed as $E_{K_1}(M_1)||E_{K_2}(M_2)$.

(a) What is the decryption algorithm? In particular, suppose we have a 64-bit ciphertext C, split into two parts C_1 and C_2, and suppose we know the 64-bit key K (whose two parts are K_1 and K_2). How do we compute M?

Solution: $M = M_1||M_2$ where $M_1 = D_{K_1}(C_1), M_2 = D_{K_2}(C_2)$. Here $D(\cdot)$ is decryption for the 32-bit block cipher.

(b) Show that this scheme is vulnerable to a known-plaintext attack. In particular, assume we know that C is the encryption of the 64-bit message M (using the 64-bit encryption algorithm, under the unknown key K). Show how to recover the 64-bit key K in a reasonable amount of time.

Solution: First, use brute force to recover K_1. We know that $C_1 = E_{K_1}(M_1)$, and we know M_1, C_1, so try all possibilities for K_1 and see which one is consistent with this equation. Next, use brute force to recover K_2, by a similar method. This requires $2^{32} + 2^{32} = 2^{33}$ trial decryptions in total, which is easily feasible.

Problem 7 ***Authentication*** **(12 points)**

Alice would like to send a message M to Bob with confidentiality and integrity. Alice and Bob share symmetric keys k_1, k_2. Bob's public key is K_B; we assume that Alice knows K_B. Below, F is AES-CMAC (a secure message authentication code) and H is SHA256 (a secure cryptographic hash).

Consider the following two schemes:

S1: Alice sends $E_{k_1}(M), F_{k_2}(M)$ to Bob
S2: Alice sends $E_{k_1}(M), H(M)$ to Bob

Here $E_{k_1}(\cdot)$ is AES-CTR mode (a secure symmetric-key encryption scheme).

(a) Which scheme is better for confidentiality, S1 or S2? Why?

Solution: S1 is better. S2 lets the attacker test a guess at M (given a guess g, compute $H(g)$; if $H(g) = H(M)$, then he can conclude his guess was correct). Therefore, if there are only a few possibilities for M (maybe it is a 4-digit PIN), S2 allows the attacker to find M.

As a special case, we also gave full credit for an answer which said that S2 is better since a MAC might potentially leak information about its message input (as explained in the lecture notes). In fact, that doesn't affect AES-CMAC—the AES-CMAC tag doesn't leak anything about the message—so for this particular question that issue doesn't actually arise, but we thought this answer demonstrated a high level of knowledge about MACs, so we gave full credit for that answer as well.

We also gave full credit for an answer which said that S1 is better since in S2, if you send the same message twice, it'll have the same hash, so an eavesdropper can tell if you send the same message twice. This answer is actually not a valid reason to prefer S1 (in fact, because a MAC is a deterministic function, both schemes suffer from this problem), but we thought that it demonstrates reasonable understanding of the topic, so we gave it full credit.

Common misconception: A MAC is not a trapdoor one-way function. There is no trapdoor. (A hash is not a trapdoor one-way function, either. Again, there is no trapdoor.) A trapdoor would mean a secret that, if you know it, lets you invert the function; but a MAC takes a long message and produces a short tag, so there is no hope of inverting it, even if you know the key.

Common misconception: Some people thought that using a hash would be secure since it is collision-resistant. This reasoning would be correct if we somehow knew that the adversary could not tamper with the value $H(M)$ that Alice sends. However, what this overlooks is that an attacker could modify both the ciphertext $E_{k_1}(M)$ *and* the corresponding hash value $H(M)$, changing the hash value to something that is consistent with the ciphertext. So, the lesson is that an active attacker can potentially change everything that's sent.

Common misconception: Some folks suggested that Bob could decrypt $F_{k_2}(M)$ to recover M, and compare it to the decryption of $E_{k_1}(M)$. Unfortunately, you can't decrypt a MAC. For instance, the input to a MAC might be a 1000-byte message, and the output is a 16-byte MAC tag, so given the MAC tag, there is no way to uniquely determine the message (not even if you know the key).

(b) Which scheme is better for integrity, S1 or S2? Why?

Solution: S1 is better. With S2, if the attacker knows M, he can modify the ciphertext to turn it into a valid encryption of M'. In particular, since E is a stream cipher, he can flip bits in $E_{k_1}(M)$ to change it to an encryption of $E_{k_1}(M')$, and then replace $H(M)$ with $H(M')$. This attack is not possible against S1, since the MAC is secure and the attacker doesn't know the MAC key k_2, so the attacker won't be able to predict $F_{k_2}(M')$.

Next, consider the following two schemes:

S3: Alice sends $E_{K_B}(M), F_{k_2}(M)$ to Bob
S4: Alice sends $E_{K_B}(M), H(M)$ to Bob

Here $E_{K_B}(\cdot)$ represents El Gamal encryption.

(c) Which scheme is better for integrity, S3 or S4? Why?

Solution: S3. With S4, anyone can forge a message to Bob; they don't need to know any secret keys.

Problem 8 ***Chosen-ciphertext security of El Gamal*** **(10 points)**

Alice is using El Gamal encryption to send messages to Bob.

Recall that the El Gamal encryption of message M is the ciphertext $C = (C_1, C_2)$, where $C_1 = g^r \bmod p$, $C_2 = M \times B^r \bmod p$, r is a random number chosen separately for each encryption, and B is Bob's public key. Bob's private key is b, and $B = g^b \bmod p$. p and g are fixed and public. To decrypt a ciphertext $C = (C_1, C_2)$, Bob computes $M = C_1^{-b} \times C_2 \bmod p$.

Suppose that, after Bob decrypts a ciphertext, if it does not look like English, Bob will send back an unencrypted email saying "What happened? Your message was corrupted" and attach the decrypted message as an attachment. Thus, Mallory can learn M, the decryption of C, for any C such that M does not look like English.

Suppose that Mallory has intercepted a ciphertext $C^* = (C_1^*, C_2^*)$ sent by Alice to Bob. Mallory knows C^* is the encryption of some English message, but Mallory doesn't know what the message says. Describe a chosen-ciphertext attack that would let Mallory learn the decryption of C^*.

Solution: Mallory can send $C = (C_1, C_2)$ to Bob, where $C_1 = C_1^*$ and $C_2 = 3C_2^* \bmod p$. Bob will decrypt this and get

$$M = C_1^{-b} \times C_2 = (C_1^*)^{-b} \times 3C_2^* = 3M^* \bmod p.$$

Now M is unlikely to look like English, so Bob will respond and Mallory will learn M. Finally Mallory can compute $M^* = M/3 \bmod p$.

An alternate solution: Mallory can send $C = (C_1^*, 1)$. The decryption will be $M = (C_1^*)^{-b} \times 1 \bmod p$, which will most likely not look like English, so Mallory learns

$(C_1^*)^{-b} \bmod p$. Now Mallory can compute M^* via the equation $M^* = (C_1^*)^{-b} \times C_2^* \bmod p$, since she knows both quantities on the right-hand side.

A common misconception: a number of people wrote something like "since we know $g^b \bmod p$, g, and p, it is easy to solve for b, and then we have the private key and can decrypt anything we want." Unfortunately, that's not right: it's not easy to solve for b. In fact, finding b from g^b, g, p is the discrete log problem, which is believed to be computationally infeasible. (If you could find b from g^b, g, p efficiently, then you could break Diffie-Hellman and El Gamal encryption. As far as we know, this is not possible.) Or, to put it another way, the function $f(x) = g^x \bmod p$ is believed to be a one-way function; so given $f(b) = g^b \bmod p$ (and g, p), there is no known feasible way to recover b. This forms part of the basis for the security of several public-key cryptosystems.