CS 161 Computer Security — Spring 2010 Paxson/Wagner — Final Solns

Problem 1. [Spoofing attacks] (18 points)

Usually, the DNS protocol runs over UDP. However, it is also possible for DNS to use TCP.

(a) Suppose you are using your laptop on an open wireless network and an attacker is within range of the wireless network, so the attacker can eavesdrop on all your traffic and inject forged packets. **Circle one** of the following that best describes the threat the attacker poses:

1. The attacker can successfully inject a spoofed DNS response if your laptop uses UDP for all of its DNS queries, but not if it uses TCP for all of its queries.
2. The attacker can successfully inject a spoofed DNS response if your laptop uses TCP for all of its DNS queries, but not if it uses UDP for all of its queries.
3. [✓ The attacker can successfully inject a spoofed DNS response if your laptop uses either TCP or UDP for its DNS queries.]
4. The attacker cannot successfully inject spoofed DNS responses.

Comment. *TCP sequence numbers don't prevent spoofing, since the attacker can eavesdrop on the TCP connection and observe the sequence numbers.*

(b) Suppose you access the Internet over a secured Ethernet network, so that the attacker cannot eavesdrop on your traffic, but the attacker can still inject forged packets. You can use either TCP or UDP for your DNS queries. Assume that the relevant TCP implementations choose Initial Sequence Number (ISNs) uniformly at random, and that the relevant DNS implementations do not implement source port randomization. Regarding Kaminsky-style "blind spoofing" of DNS replies, **circle one** of the following that best describes the threat the attacker poses:

1. [✓ When you use TCP for your queries you are safer (harder to attack) than when using UDP.]
2. When you use UDP for your queries you are safer (harder to attack) than when using TCP.
3. You are equally vulnerable to the attack whether you use UDP or TCP.
4. In this scenario, you are not vulnerable to the attacker regardless of whether you use UDP or TCP.

Comment. *In Kaminsky-style "blind spoofing" attacks, if you're using UDP, the forged packet has to contain the right 16-bit DNS ID. If you're using TCP, the forged packet has to contain the right 16-bit DNS ID and the right 32-bit TCP sequence number. The latter is much harder: it has a much lower success probability ($1/2^{48}$ vs. $1/2^{16}$).*

(c) Suppose we could deploy a mechanism that would ensure IP source addresses always correspond to the actual sender of a packet—in other words, suppose it is impossible for an attacker to spoof source addresses. **Circle all** of the following threats that this mechanism would *completely eliminate*. By "eliminate a threat", we mean that the anti-spoofing mechanism would suffice to prevent exploitation of the threat, without any additional mechanisms or assumptions.

1. Buffer overflow attacks
2. Cross-site request forgery (CSRF) attacks
3. TCP SYN flooding
4. [✓ TCP RST injection]
5. Spam
6. None of the above

Comment. *It doesn't completely prevent the others: the other kinds of attacks can still be carried out, without ever sending a single packet with a spoofed source address.*

(d) Again suppose we could deploy a mechanism that would ensure IP source addresses always correspond to the actual sender of a packet. **Circle all** of the following threats for which this mechanism would eliminate at least *some* common instances of the attack but not *all* instances. "Eliminate an attack instance" refers to preventing that attack instance from succeeding, without any additional mechanisms or assumptions.

1. Buffer overflow attacks
2. Cross-site request forgery (CSRF) attacks
3. [✓ TCP SYN flooding]
4. TCP RST injection
5. Spam
6. None of the above

Comment. *The most common form of TCP SYN flooding involves sending many SYN packets, each with a different spoofed source address (to make it hard to block the malicious SYN packets).*

Problem 2. [Reasoning about memory-safety] (24 points)

Consider the following C code:

```
void delescapes(char s[], int n) {
    int i=0, j=0;
    while (j < n) {
        if (s[j] == '%') {
            j=j+3;
        } else {
            s[i] = s[j];
            i=i+1; j=j+1;
        }
    }
}
```

We'd like to know the conditions under which `delescapes()` is memory-safe, and then prove it.

On the next page, you can find the same code again, but with blank spaces that you need to fill in. Find the blank space labelled `requires:` on the next page, and fill it in with the precondition that's required for `delescapes()` to be memory-safe. (If several preconditions are valid, you should list the most general precondition under which it is memory-safe.)

Also, on the next page fill in the three blanks inside `delescapes()` with invariants, so that (1) each invariant is guaranteed to be true whenever that point in the code is reached, assuming that all of `delescapes()`'s callers respect the precondition that you identified, and (2) your invariants suffice to prove that `delescapes()` is memory-safe, assuming that it is called with an argument that satisfies the precondition that you identified.

Keep in mind that, as emphasized in the last homework, invariants should be self-contained and state all facts that are needed for a proof of memory-safety.

You may ignore the possibility of NULL dereference errors for this question.

You may use the notation `size(s)` to refer to the amount of memory allocated for `s`.

Here is the same C code again, this time with space for you to fill in the precondition and three invariants. Remember, fill in all blanks.

```
/* requires: n <= min(size(s), MAXINT-3) */
void delescapes(char s[], int n) {
    int i=0, j=0;
    while (j < n) {
        /* 0 <= i <= j < n <= min(size(s), MAXINT-3) */
        if (s[j] == '%') {
            j=j+3;
            /* 0 <= i <= j && n <= min(size(s), MAXINT-3) */
        } else {
            s[i] = s[j];
            i=i+1; j=j+1;
            /* 0 <= i <= j <= n <= min(size(s), MAXINT-3) */
        }
    }
}
```

Comment. *There are several other acceptable variations. For instance, you could have written* `1 <= i <= ...` *for the last invariant instead of* `0 <= i <= ...`*.*

A subtle point: To avoid integer overflow, the precondition must include the condition `n <= MAXINT-3`*. Without this precondition,* `delescapes()` *is not necessarily memory-safe. For example, consider a very long string* `s`*, one that is* `MAXINT` *characters long, and suppose* `n = MAXINT`*. Suppose moreover that the string ends with a* `'%'` *character. Then when the loop is processing the last character of the string, namely when* `j = MAXINT-1`*, we'll add 3 to* `j`*, causing* `j` *to wrap around and become negative. Subsequent iterations of the loop will now attempt to dereference* `s[j]`*, where the array index* `j` *is negative. This is an array out-of-bounds error. This problematic situation can be avoided as long as* `n <= MAXINT-3`*. (Here* `MAXINT` *is the largest positive number representable in an* `int`*, e.g.,* $2^{31}-1$ *on a typical 32-bit platform.) This is admittedly a subtle point, so if you missed this point, don't feel too bad. You will receive significant partial credit if you overlooked this and your answer was otherwise correct.*

Problem 3. [Espionage] (25 points)

For this problem assume the existence of a super-top-secret *Incredibly Valuable Document* (IVD) belonging to a business competitor. You are desperate to read the IVD and are prepared to undertake dubious measures to do so. The IVD is 100 KB (kilobytes) in size and you already have a copy of it encrypted using a 128-bit AES key. You have 3 avenues available for reading the document:

- **Buy bots**, at a cost of \$1 per 2^{10} bots. Each bot can brute-force 2^{40} keys per week. (FYI, this is about 1.8 million keys per second.) You can buy up to 2^{24} ($\approx$ 17 million) bots and then, alas, the underground market has no more to offer you.
- **Bribe** an employee of the competitor who has access to the key used to encrypt the document. The employee is willing to leak out to you, via a covert channel, 4 bits of the key each week. Because you blew it and let on to the employee how desperate you are, each set of 4 bits will cost you \$100,000.
- Employ an incredibly sophisticated **side channel attack** that can remotely monitor the power lines going to the competitor's machine room and from their tiny fluctuations infer 1 KB of the document each week. (Each week you obtain a new 1 KB chunk.) This scheme, however, costs \$1,000,000 up front for lasers and superconducting magnets and cool touch-screen technology.

You can employ any or all of these as you see fit. You may use a combination of these methods.

(a) Suppose that what's most important is recovering the document as quickly as possible, in terms of *guaranteed running time*. Which approach will do so, roughly how long do you expect it to take (in weeks), and how much will it wind up costing?

Answer. *Quickest approach: Bribe the employee for 16 weeks, learning 64 bits of the key. Then, buy 2^{24} bots and use them to brute-force the remaining 64 bits of key.*

Approximately how many weeks it will take: 17 weeks.

Approximately how much it will cost: \$1,616,384. (\$1,600,000 for the bribes and \$16,384 for the bots)

Comment. *The hybrid approach (a combination of bribery and brute force) is faster than any single strategy. On its own, brute force would take 2^{64} weeks, if you buy up all available bots; that's longer than the expected lifetime of the solar system. Bribery alone would take 32 weeks, which is the fastest if we ignore hybrid strategies but is slower than the best hybrid strategy.*

(b) Suppose that what's most important is recovering the document as cheaply as possible, providing you get your hands on it within 5 years (260 weeks). Which approach will do so, roughly how much do you expect it to cost, and about how long will it take (in weeks)?

Answer. *Cheapest approach: Use the side channel.*

Approximately how much it will cost: \$1,000,000.

Approximately how many weeks it will take: 100 weeks.

Comment. *If we don't use the side channel, and use a combination of bribery and brute force, the cheapest method is to bribe the employee for 15 weeks, then brute-force for 16 weeks to recover the remaining 68 bits of key. This costs \$1,500,000 for the bribes alone, and thus is not cost-competitive. (If we bribe for only 14 weeks, then brute-force will take 256 weeks, for a total of 270 weeks, which exceeds our time limit.) So no combination of bribery and brute force will be cheaper than the side channel, within the 260-week time limit.*

There is no advantage to combining the side channel attack with bribery or brute force.

Problem 4. [Detecting attacks] (30 points)

You work for a company that sells intrusion detection systems. When you start at the company, the core technology is a signature-based scheme we'll call *S*. A little later the company acquires a competitor whose core technology is an anomaly-based scheme we'll call *A*.

Suppose that *S* is a network-based scheme that works by passively analyzing individual UDP and TCP packets. Suppose that *A* is a host-based scheme that is a component of the browser that processes and analyzes individual URLs before they're loaded by the browser. Scheme *S* operates in a stateless fashion and scheme *A* maintains state regarding URLs it has previously analyzed.

(a) For each of the following, circle your answer or answers.

1. With regard to the general properties of different types of detectors, **circle all** of the following that are correct:

   i. To use a signature-based approach like scheme *S* at a new site first requires access to logs of the site's historical activity in order to train the detector.

   ii. [✓ It is possible to design a signature-based detector that has no false negatives.]

   iii. Specification-based approaches work well for detecting known attacks but do not work well for detecting novel attacks.

   iv. [✓ One appealing property of network-based detectors is that they can provide easier management than host-based detectors.]

   v. An attraction of behavioral approaches is that they are especially well-suited to prevent initial system compromises.

   vi. None of the above.

   Comment. *To achieve no false negatives, simply alert on every URL you see.*

2. Your company wants to deploy a new product that provides intrusion prevention functionality, building upon either *S* or *A* as a foundation. The best candidate scheme for this is (**circle just one**):

   i. Scheme *S*, because signature-based approaches have lower false negative rates than anomaly-based approaches.

   ii. Scheme *S*, because signature-based approaches work in real time and anomaly-based approaches do not.

   iii. Scheme *A*, because anomaly-based approaches have lower false positive rates than signature-based approaches.

   iv. Scheme *A*, because anomaly-based approaches require less state than signature-based approaches.

   v. [✓ It is not clear without additional information which of schemes *S* or *A* would work better for intrusion prevention.]

   Comment. *The key aspect this problem attempts to get at is that many characteristics of a detection algorithm (false positive vs. false negative rates, operating in real-time vs. post facto, stateful vs. stateless operation) are independent of whether the detection is signature-based or anomaly-based. In addition, when choosing a detection algorithm one needs to assess the cost of incurring false positives versus incurring false negatives.*

3. As your company becomes more successful you grow concerned that attackers will try to evade your detectors. **Circle all** of the following that are correct:

   i. [✓ Scheme *S* is vulnerable to evasion by attackers who can manipulate the order and timing of the packets they send.]

ii. Scheme A is vulnerable to evasion by attackers who can force their traffic to be sent using fragmented packets.

iii. An attacker can more easily try to exhaust the memory used by scheme S than the memory used by scheme A.

iv. For analyzing unencrypted HTTP traffic, scheme A is more vulnerable to evasion by attackers that employ URL hex-escape encodings than is scheme S.

v. [✓ Given how the different schemes operate, scheme A is likely better suited for resisting evasion by imposing a canonical form (“normalization”) on the traffic it analyzes than is scheme S.]

vi. None of the above.

Comment. *Scheme S works on individual packets in a stateless fashion. This makes it easy for an attacker to split an attack across multiple packets so that S will miss it.*

Because scheme A operates inside the browser, any traffic it analyzes has already been completely reassembled and disambiguated by the operating system, so A is not vulnerable to evasion through fragmented packets.

For iii., the opposite is true, because the problem states that S is stateless while A is stateful.

Because scheme A operates inside the browser, it has at least as complete a view of the URL processing as does scheme S, which operates separate from the browser. Therefore, A is less vulnerable to evasion through URL encodings.

Normalization requires being able to alter the traffic that a monitor inspects. The problem states that scheme S works passively, while scheme A is part of the brower's processing sequence. This makes A able to alter traffic while S cannot.

(b) Your company decides to build a hybrid scheme for detecting malicious URLs. The hybrid scheme works by combining scheme S and scheme A, running both in parallel on the same traffic. The combination could be done in one of two ways. Scheme H_E would generate an alert if for a given network connection *either* scheme S or scheme A generates an alert. Scheme H_B would generate an alert only if *both* scheme S and scheme A generate an alert for the same connection. (Assume that there is only one URL in each network connection.)

Assuming that decisions made by S and A are well-modeled as independent processes, and ignoring any concerns regarding evasion, which of the following statements regarding the hybrid scheme are correct? **Circle all** that apply.

1. H_E will result in a lower false positive rate than scheme H_B.
2. [✓ H_E will result in a lower false negative rate than scheme H_B.]
3. [✓ H_E will be likely to detect a larger number of known attacks than H_B.]
4. [✓ H_E will be likely to detect a larger number of novel attacks than H_B.]
5. None of the above.

Comment. *The key insight to answering this problem is that the alerts produced by H_E will be a superset of those produced by H_B, since S and A are independent processes.*

(c) If deploying the hybrid scheme in a new environment, and under the same assumptions as in part (c), is one of H_E and H_B clearly better? If yes, explain which would be your choice and why. If not, explain why there is no clear choice.

Answer. *No, it's not clear which choice is better. Without knowing how the cost of false positives compares to the cost of false negatives, it's impossible to say which choice will be better.*

Problem 5. [Viruses and Worms] (25 points)

(a) Which of the following is true of viruses/worms? **Circle all that apply**.

1. Polymorphic viruses are harder to detect with signature-based techniques than metamorphic viruses.
2. You can prevent metamorphic viruses from infecting your systems by making your disks read-only after bootup.
3. Metamorphic viruses require public-key cryptography.
4. [✓ You can write a worm that uses multiple different techniques to spread.]
5. None of the above.

(b) Which of the following is true of worms? **Circle all that apply**.

1. Worms were first invented less than 10 years ago, with the appearance of Code Red.
2. Recovering from a worm infection that compromises a Linux administrator account requires rebuilding the system from its original source code.
3. [✓ There have been Internet worm outbreaks that infected more than 1,000,000 systems.]
4. There have been Internet worm outbreaks that infected more than 250,000 systems in under 5 seconds.
5. [✓ For a worm that randomly scans Internet addresses and has a constant (per infected host) scanning rate, the larger the vulnerable population the more quickly the worm spreads.]
6. None of the above.

Comment. *Recovering from an infection may require reinstalling the OS, but the OS doesn't need to be recompiled from source code (and recompiling may not be sufficient, if you use a compiler that the worm has tampered with).*

Worms were invented over 10 years ago. For instance, the Morris worm dates back to 1988, significantly predating Code Red.

There hasn't been any Internet worm that infected 250,000 systems in under 5 seconds. The Slammer worm is arguably the fastest propagating Internet worm, but it infected only 75,000 machines and took a number of minutes to do so.

Several worms have infected over 1 million machines. It was mentioned in lecture that one worm infected over 15 million machines.

The last item regarding random-scanning worms was mis-phrased. It was meant to get at the fact that such worms spread faster across larger vulnerable populations. However, the reason they do is that for random-scanning worms, the contact rate β grows larger with the size of the vulnerable population. Thus, the problem's description of the contact rate being "constant" is quite confusing. Accordingly, we gave credit for this item whether circled or not.

(c) Suppose you have a technology available that will prevent any buffer overflow attack. If you deploy it everywhere, which of the following best describes its effectiveness? **Circle just one.**

1. It would prevent all types of worms from propagating.

2. It would slow down the propagation of all types of worms but not fully prevent the propagation of any.
3. [✓ It would prevent the propagation of some types of worms, but not all types.]
4. It would not help in preventing worms from propagating, but would slow down the propagation of some types of worms.
5. It would not help in preventing worms from propagating nor in slowing down their propagation.

Comment. *Some worms spread by exploiting buffer overrun vulnerabilities. If this is the only means of propagation for a particular worm, then this technology will prevent that worm from propagating.*

(d) A random-scanning worm spreads (**circle just one**):

1. with linear speed until it reaches critical mass, after which it propagates at an exponentially increasing rate.
2. [✓ exponentially quickly at the beginning, but with the rate decreasing as more and more systems are infected.]
3. at a quadratically increasing rate throughout the worm's propagation.
4. at an exponentially increasing rate until all susceptibles are infected.

Comment. *The number of infected hosts at any given time is well-modelled by a logistic function. Early in the propagation, this is well-approximated by an exponential function. Once the worm has infected a non-trivial fraction of the vulnerable population, the rate of spread decreases.*

Problem 6. [Crypto] (32 points)

This problem tests your understanding of how to use cryptographic algorithms. Alice and Bob conduct business extensively over the Internet, and they would like to be able to enter into binding contracts with other parties located around the world. To save the environment, they'd like to do it entirely electronically, over the Internet. Therefore, we'd like some cryptographic way that each party can agree to the terms of the contract. If Alice and Bob are contemplating a contract, Alice should be able to tell whether Bob has agreed to the contract, and similarly Bob should be able to tell whether Alice has agreed to the contract.

Once Alice and Bob have agreed to the terms of the contract, neither should be able to back out. If one party fails to live up to his/her obligations, or there is any dispute, we assume that the injured party will bring a lawsuit. In case of a lawsuit, it should be possible for Alice to convince the judge that Bob agreed to the terms of the contract (by showing the judge the messages she received from Bob). Similarly, it should be possible for Bob to convince the judge that Alice agreed to the contract. It should never be necessary for any party to give their private key to the judge, but it is OK to give the judge any session keys that are relevant.

The scheme should be secure against attackers with the ability to intercept and/or modify packets sent electronically, and with the ability to inject forged packets. There should be no way for an attacker to fool Alice into thinking that Bob has accepted any contract that he did not actually accept, and no way to fool Bob into thinking that Alice has accepted any contract that she did not actually accept. Also, in the event of a lawsuit, there should be no way for Alice to fool the judge into concluding that Bob accepted the contract when he actually didn't (even if Alice colludes with the attacker), and no way for Bob to fool the judge into concluding that Alice accepted the contract when she actually didn't (even if he colludes with the attacker). Let's look at several possible cryptographic protocols for this.

Assumptions: You can assume that Alice has a public key K_A and a matching private key K_A^{-1}; Bob has a public key K_B and private key K_B^{-1}; and the court has a public key K_C and a private key K_C^{-1}. Assume that everyone knows everyone else's public key, with no possibility of spoofed public keys (e.g., Alice knows Bob's public key and the court's public key, and so on). The parties do not share their private key with anyone. Each party has his/her own personal computer, which is not shared, is adequately protected from compromise, and can be assumed free of malware. You may assume that all encryption, MAC, and digital signature algorithms are secure against chosen-plaintext attack and are implemented properly. Let T denote the text of the contract. T will include the identities of the parties and all the terms of the contract. Alice and Bob already know T. There is no need to prevent the attacker from learning T. At least one of the schemes below is OK.

Instructions: For each scheme listed below, circle "OK" if the scheme meets the requirements above, or "NOT" if the scheme is does not meet the requirements above (e.g., it is insecure). Each part is worth 4 points; you will receive 4 points for a correct answer, 2 points if it is left blank, and 0 points for an incorrect answer.

(a) OK or [✓ NOT]: Alice sends $E_{K_B}(T)$ to Bob. Bob sends $E_{K_A}(T)$ to Alice.

> **Comment.** *Since anyone can create the ciphertext $E_{K_B}(T)$, Bob has no way of telling whether Alice sent the ciphertext $E_{K_B}(T)$ or some third party did. Also, Bob has no way of convincing the judge that Alice generated this ciphertext, as opposed to someone else (or even Bob himself).*

(b) OK or [✓ NOT]: Alice sends $E_{K_C}(T)$ to Bob. Bob sends $E_{K_C}(T)$ to Alice.

> **Comment.** *Bob has no way of determining what message Alice encrypted, and whether it corresponds to the terms T that he is expected. If Bob does show $E_{K_C}(T)$ to the judge, the judge has no way of knowing who generated it (Alice? Bob? someone else entirely?).*

(c) OK or [✓ NOT]: Alice sends $E_{K_B}(T), E_{K_C}(T)$ to Bob. Bob sends $E_{K_A}(T), E_{K_C}(T)$ to Alice.

> **Comment.** *Bob still cannot determine who created these ciphertexts. Also, Bob has no way to check whether the two ciphertexts decrypt to the same thing.*

(d) [✓ OK] or NOT: Alice sends $\text{Sign}_{K_A^{-1}}(T)$ to Bob. Bob sends $\text{Sign}_{K_B^{-1}}(T)$ to Alice.

> **Comment.** *Bob can confirm that Alice agreed to the terms of the contract (no one else can generate the signature). Bob can prove to the judge that Alice indicated her assent (not even Bob could have generated the signature).*

(e) [✓ OK] or NOT: Alice sends $\text{Sign}_{K_A^{-1}}(H(T))$ to Bob, where H is a cryptographic hash function. Bob sends $\text{Sign}_{K_B^{-1}}(H(T))$ to Alice.

> **Comment.** *Since H is a cryptographic hash function, it is infeasible to find collisions. This means that Alice cannot change her mind and later claim that she was agreeing to some other terms T' (that would only be possible if Alice could find T, T' such that $T \neq T'$ and $H(T) = H(T')$, but for a cryptographic hash function, this is infeasible).*

(f) OK or [✓ NOT]: Alice picks a random session key k and sends $k, E_k(T), \text{Sign}_{K_A^{-1}}(k), \text{Sign}_{K_A^{-1}}(E_k(T))$ to Bob. Bob picks a random session key k' and sends $k', E_{k'}(T), \text{Sign}_{K_B^{-1}}(k'), \text{Sign}_{K_B^{-1}}(E_{k'}(T))$ to Alice.

Comment. *Suppose Alice and Bob agree on two contracts, T_1 and T_2. Bob receives $k_1, E_{k_1}(T_1), Sign_{K_A^{-1}}(k_1), Sign_{K_A^{-1}}(E_{k_1}(T_1))$ and $k_2, E_{k_2}(T_2), Sign_{K_A^{-1}}(k_2), Sign_{K_A^{-1}}(E_{k_2}(T_2))$. Bob can now go to the judge and falsely claim that Alice agreed to a contract with terms $T_3 = D_{k_1}(E_{k_2}(T_2))$. When the judge asks for evidence, Bob can claim Alice sent him the message $k_1, E_{k_2}(T_2), Sign_{K_A^{-1}}(k_1), Sign_{K_A^{-1}}(E_{k_2}(T_2))$. The signatures will all be valid, and when the judge decrypts to obtain the terms of the contract, he will obtain T_3. So the judge will conclude that Alice agreed to these terms, when in fact she did no such thing. This violates the security requirements.*

Similarly, if Alice and Bob agree on two contracts, with terms T_1 and T_2, then an eavesdropper can construct a message that will fool Bob into thinking that Alice agreed to a contract with terms $T_3 = D_{k_1}(E_{k_2}(T_2))$. This too violates the security requirements.

In both examples, the bogus terms T_3 will most likely be random, meaningless gibberish, so this might seem unlikely to cause serious damage, at least if both participants are humans. Nonetheless, it violates the security requirements. (Moreover, if Alice and Bob agree on sufficiently many contracts, it may be possible to find some pair which can be combined to yield a meaningful and harmful contract that neither agreed upon. If Alice and Bob are software programs that act automatically on these contracts, without any human involvement, then the threat might increase further.)

(g) [✓ OK] or NOT: Alice picks a random session key k and sends $U, \text{Sign}_{K_A^{-1}}(U)$ to Bob, where $U = (k, E_k(T))$ (U is the concatenation of k and $E_k(T)$). Bob picks a random session key k' and sends $U', \text{Sign}_{K_B^{-1}}(U')$ to Alice, where $U' = (k', E_{k'}(T))$.

Comment. *The value U is a funny encoding of the terms T. In particular, $U = f(T)$, for some invertible function f, the details of which are not important. Since f is invertible, given U it is possible to uniquely recover the terms T. Therefore, once Alice sends her message to Bob, she is committed: both Bob and the judge can uniquely identify the terms that she agreed to.*

(h) OK or [✓ NOT]: Alice picks a random session key k and sends $\text{MAC}_k(T), E_{K_B}(V)$ to Bob, where $V = (k, \text{Sign}_{K_A^{-1}}(k))$. Bob picks a random session key k' and sends $\text{MAC}_{k'}(T), E_{K_A}(V')$ to Alice, where $V' = (k', \text{Sign}_{K_B^{-1}}(k'))$.

Comment. *Once Bob receives a message from Alice agreeing to the terms T, Bob can go to the judge and claim that Alice agreed to some other terms T' (where $T \neq T'$). When the judge asks for evidence, Bob can claim that Alice sent him the message $MAC_k(T'), E_{K_B}(V)$, where $V = (k, Sign_{K_A^{-1}}(k))$. Since Bob knows the session key k, he can compute the MAC on anything he wants, including on T'.*

Problem 7. [Password hashing] (6 points)

Suppose you are reviewing email encryption software used by millions of people. You discover it encrypts every message under a 128-bit key K derived as $K = F(P)$, where P is a password chosen and entered by the user. The encrypted message is then broadcast over an insecure network. However, you have not reverse-engineered the algorithm for F yet, and all you know is that F is a deterministic unkeyed function. What can you conclude about the security of this key generation method? **Circle just one** (the best answer).

1. If F is not publicly documented anywhere, this is an excellent design, because it ensures that no one will ever be able to learn the encryption key K.

2. This will be secure as long as F has the property that all 2^{128} possible keys can occur as the output of F (i.e., for every $K \in \{0,1\}^{128}$ there exists P such that $K = F(P)$). F does not need to be secret; it is OK if F is publicly documented.

3. This will be secure as long as F is a cryptographic hash function. F does not need to be secret; it is OK if F is publicly documented. However, it is not sufficient to meet the conditions of answer 2: i.e., it is not sufficient for F to merely have the property that all 2^{128} possible keys can occur as the output of F.

4. This design is problematic no matter how F is chosen, because the key K will most likely not have enough entropy. It will be problematic even if F is a cryptographic hash function.

Comment. *Most users choose passwords with fairly low entropy: far less than 128 bits of entropy. In particular, it is often possible to recover many users' passwords by trying many dictionary words and slight variations on dictionary words. Since F is a deterministic function, the entropy of the key $F(P)$ is no greater than that of the password P.*

Comment. *If passwords did have sufficient entropy, then answer 3 would suffice. A cryptographic hash function guarantees to preserve essentially all the entropy in its input, or 128 bits, whichever is less. If the input P has enough entropy to be unpredictable and F is a cryptographic hash function, then $F(P)$ looks indistinguishable from random (indistinguishable from a value chosen uniformly at random). However, in practice passwords typically do not have sufficient entropy.*

Answer 2 would not be acceptable, even if passwords did have enough entropy. There are functions F where all 2^{128} outputs are possible, but not all are equally likely. (For instance, consider the function F whose output is all zeros, except for 256-bit inputs whose first 128 bits are zero; for those inputs, the output of F is equal to the last 128 bits of its input. Such a function satisfies the conditions of answer 2, but would be no good for generating cryptographic keys.)

Problem 8. [Denial of service] (20 points)

`BrowserTest.com` is a useful new web service that helps you compare how a web site looks when viewed with Firefox vs. how it looks when viewed with Internet Explorer.

Here's how it works. If you visit a URL like `http://browsertest.com/?u=http://cnn.com/`, the BrowserTest server starts a Firefox process, loads `http://cnn.com/` in Firefox, and takes a screenshot of the resulting Firefox window. In parallel, it starts Internet Explorer, loads `http://cnn.com/` in Internet Explorer, and takes a screenshot of the resulting Internet Explorer window. When both of these two parallel steps complete, the BrowserTest server responds to the original request with a HTML document containing both screenshots. (Note that the process of handling this HTTP request from the user causes the BrowserTest server to issue two separate HTTP requests to `cnn.com`, one for each browser.) The BrowserTest server is willing to accept any URL you specify; it treats everything after the `?u=` as a URL and loads that URL in each browser. Thus, the BrowserTest site is very general and useful for testing how portable a website is.

Explain how an attacker could mount a serious denial-of-service attack against the BrowserTest server simply by visiting a single carefully chosen URL. Show the malicious URL that can cause so much havoc. You may assume the BrowserTest folks haven't taken any special precautions against denial-of-service.

Answer. $\texttt{http://browsertest.com/?u=}\underbrace{\texttt{http://browsertest.com/?u=}}_{\text{repeat many times}}\cdots$

With n repetitions, BrowserTest will make 2^n HTTP requests. If n is large, this will likely overload the BrowserTest server.

Comment. *Your answer should not assume that BrowserTest's server has XSS vulnerabilities.*

Problem 9. [Terminology] (20 points)

For each of the numbered concepts given below, list the term that best applies:

Application proxy	Leap-of-faith authentication	Same origin policy
Browser-in-browser	Least privilege	Security perimeter
Bulletproof hosting	Logic bomb	Security-by-obscurity
Click-jacking	Man-in-the-middle	Separation of responsibility
Command injection	Onion router	Side channel
Complete mediation	Opportunistic ack'ing	Stored XSS
CSRF	Ransomware	Third-party cookie
Default deny	Redaction	TOCTTOU vulnerability
Defense in depth	Reference monitor	Trusted path
Directory/path traversal	Reflected XSS	Tunneling
Dongle	Remanence	VPN
Homograph attack	RST injection	Warez
Inband signaling	Rubber hose cryptanalysis	Watermarking

Not all terms are used. No term is used twice.

1. An approach to copy protection based on requiring the user to plug into their computer a specialized hardware device

   Answer. *Dongle*

2. A component in some anonymity systems that works by forwarding traffic that has been multiply encrypted using different keys

   Answer. *Onion router*

3. An alternative to PKI whereby the first time a user connects to a service they simply accept whatever public key the server offers and assume it is likely to be correct

   Answer. *Leap-of-faith authentication*

4. A web attack that exploits the generality of the file system of a server hosting web content in order to access files that the operator did not intend to make accessible

   Answer. *Directory/path traversal*

5. A web attack based on embedding a script in a URL such that when a web server processes the URL, its reply includes the script within it

   Answer. *Reflected XSS*

6. An attack whereby an adversary inserts themselves into communication between two parties without their knowledge, enabling the adversary to inspect and modify the communication

   Answer. *Man-in-the-middle*

7. A service that supports keeping Internet servers operational in the presence of complaints and "take-down" demands

   Answer. *Bulletproof hosting*

8. An attack whereby a TCP receiver spurs the host sending to it to transmit more quickly

   Answer. *Opportunistic ack'ing*

9. Overcoming the use of cryptography by threatening the person who holds the secret key rather than attempting to break the cryptographic mechanisms

   Answer. *Rubber hose cryptanalysis*

10. The problem of data persisting even though the user instructed their system to delete it

    Answer. *Remanence*

11. A web attack based on confusing users regarding what URL their browser displays by employing characters that look the same, such as the digit one ("1") for an an ell ("l")

    Answer. *Homograph attack*

12. The notion of gaining more robust security by employing multiple mechanisms of different types

    Answer. *Defense in depth*

13. The notion that one can gain more robust security by ensuring that system components cannot acquire capabilities they do not need for their operation

    Answer. *Least privilege*

14. A system component that securely mediates all access to a given object

    Answer. *Reference monitor*

15. An attack based on confusing the user about which browser window or frame is receiving their mouse events

    Answer. *Click-jacking or Browser-in-browser*

16. The notion that when enforcing access control policies, one must ensure that every access to every object is checked

    Answer. *Complete mediation*

17. Communicating control information using the same channel as is used for communication data

    Answer. *Inband signaling*

18. A web attack based on (a) predicting a URL that when sent to a server instructs the server to take actions on behalf of the user whose browser sends the URL, and (b) causing the victim's browser to send that URL to the server

    Answer. *CSRF*

19. A scheme to enable detection of copying of digital content by robustly embedding a distinct pattern in the content

    Answer. *Watermarking*

20. A software flaw that occurs when a program tests for permission to access an object separately from later actually accessing the object

    Answer. *TOCTTOU vulnerability*