CS 155
Homework 2

Rafael Moreno Ferrer
SUID#: 05490330

Problem 1. *(HTTPS vs. Signatures)*

Suppose a software company widgets.com sells a product P and wants to distribute a software update D. The company wants to ensure that its clients only install software updates published by the company. They decide to use the following approach: The company places D on its web server and designs the software P to periodically check this server for updates over HTTPS.

a) The company decides to buy a public key certificate for its web server from a reputable CA. Explain what checks P should apply to the server's certificate to defeat a network attacker. Is your design vulnerable to ssl strip?

Let S be the web server and C be the reputable CA. In my design P always makes sure that when it connects to S it is always over SSL (HTTPS). This would be a hardcoded check in the source code for P. This idea is similar to the prepopulated HSTS databases that browsers ship with, which make sure that a browser will never connect to certain websites in the clear. As we saw in class, this design (along with the following checks on S's certificate) defeats ssl-strip since the attack requires P to accept communicating with S over the clear.

Also in my design P will ship with C's public key baked in into its CA public key database. When P gets S's certificate $Cert$ in the Server-Hello HTTP response it will do the following:

1) Make sure $Cert$ is signed by a trusted CA (in this case C) and decrypt $Cert$ with C's public key (baked into P)
2) Check certificate is not expired (see $Cert$'s expiry date)
3) Check certificate has not been revoked by a CRL
4) Check that the CommonName in $Cert$ matches the domain name of S

If all of these checks pass then P will go ahead and continue with the client-key-exchange phase of the HTTPS connection setup.

b) How would you design the program P and the company's web server so that the update is secure against a network attacker, but there is no need to buy a certificate from a public CA? Your design should use an HTTPS web server as before.

In this alternative design, the program P is shipped with S's public key PK_S baked in into its CA public key database (and thus effectively considers S to be a reputable CA). S will create a self-signed certificate $Cert'$ with the same information I_S as the certificate in part (a) (expiry date, CommonName for S, **S's public key** PK_S, etc.) and self-sign it with its own private key SK_S to produce $Cert' = E_{SK_S}(I_S)$[1]. The HTTPS setup protocol will then unfold as follows:

1) P initiates a Client Hello message to S (P still is hardcoded to only open HTTPS connections to S)
2) S responds with Server Hello message and its self-signed certificate $Cert' = E_{SK_S}(I_S)$
3) P receives $Cert'$ and verifies that it came from a trusted CA, in this case S (since S's public key is baked in as a trusted CA public key). P decrypts $Cert'$ with PK_S to get $D_{PK_S}(Cert') = D_{PK_S}(E_{SK_S}(I_S)) = I_S$.
4) P does the checks listed in part (a) and proceeds only if none fail.
5) P generates a random key k, encrypts it with the public key it got from I_S which happens to be PK_S and sends $E_{PK_S}(k)$
6) S receives the message and decrypts using its private key to get $D_{SK_S}(E_{PK_S}(k)) = k$.
7) Connection setup is successfully finished and both parties proceed to communicate using symmetric key crypto with shared key k.

This approach is also secure against ssl-strip since P is also hardcoded never to connect to S unless its over HTTPS and defeats the network attacker since he cannot forge a trusted certificate since he does not possess SK_S.

[1] E_{SK} denotes the encryption function using the secret key SK and D_{PK} denotes the inverse decryption function using the corresponding public key PK.

c) Later on engineers at the company proposed the following very different design: Sign D using a widgets.com private key to obtain a signature s and then distribute (s,D) in the clear to all customers. The corresponding public key is embedded in the program P. The following two questions compare this new design to the design used in part (a):

i) If we want to distribute the patch D using a content distribution network like BitTorent, which of the two designs should we use? Explain why.

Let PK_W and SK_W be widget.com's public an private keys.

We should use the alternative design of distributing $(s, D) = (E_{SK_W}(D), D)$ and baking widgets.com public key PK_W into P.

Notice that in this situation all we care about is **a)** that P can be certain that the content D was produced by widgets.com and **b)** to ensure the integrity of D. First, note that we get these two properties by distributing (s, D) and having P verify that they "match". If P received (s', D'), in order to verify D' originated from widgets.com and has integrity it needs to decrypt s' with PK_W and check it matches D' bit by bit. If $s' = E_{SK_W}(D')$ then

$$D_{PK_W}(s') = D_{PK_W}(E_{SK_W}(D')) = D'$$

An attacker would not be able to produce a pair (s', D') that matched since he does not possess SK_W, and if he corrupted D' then the decryption of s' would not match D'. Hence if $s' = E_{SK_W}(D')$ then with high probability P can trust D' came from widgets.com and has integrity. Therefore it must be the case that $D' = D$, i.e. P got the original patch produced by widgets.com.

Next, note that we care only about ascertaining the identity of the entity that *produced* D and not the identity of the entity that *distributed* D, i.e. we do not care about who delivers D to P as long as the two conditions above are met. In particular we do not require that the parties that distribute and deliver (s, D) to the end program be trusted. This is why a CDN of not necessarily trusted peers that transmits (s, D) over the clear meets our needs.

The first approach is unnecessary here since if P were to establish HTTPS connections to talk to the peers in the CDN then effectively we would be requiring some authentication of the peers to P. As we said we do not care who the distributors are as long as (a) and (b) above are met so we do not need to validate the identities of the peers. Not to mention the added practical disadvantages of using HTTPS for communicating with the CDN: if we used the approach for part a) all peers would need to get certificates from a trusted CA which would be expensive, time consuming, and against the nature of peer to peer networks, and if we used the approach from part b) we would need to put all the public keys of the peers into P, which would defeat the flexibility of CDNs since we would need to know all trusted peers beforehand. Both of these cases would incur the costs of per-connection crypto computations (see next point).

ii) List the most time consuming crypto operations that the widgets.com web server does in each of the two designs? It suffices to only focus on pubic-key operations such as public-key encryption/decryption and digital signature generation /verification. Which of the two designs is more efficient?

The alternative design of distributing (s, D) is more resource efficient. Basically in this approach there is a single expensive crypto operation being done offline while in the first approach there are crypto operations being incurred per every connection.

In the approach that distributes (s, D) over the clear the most time consuming crypto operation being done by the widgets.com web server is that of **signing D to produce $s = E_{SK_W}(D)$**. This cost is payed offline, upfront, and only once before the patch is distributed and never incurred again afterwards.

In contrast, in the HTTPS approach there is a lot more crypto work done. Since we are establishing encrypted connections between every instance of P and the widgets.com web server we need to incur crypto operations per connection (which grows linearly with the number of downloads). In particular if we see the HTTP preamble/setup from part (b) we will see that in step 6 (client-key-exchange phase) S receives an encrypted message $m = E_{PK_S}(k)$ containing the random symmetric key generated by P and S **has to perform $D_{SK_S}(m)$ to decrypt it**. This, in addition to the symmetric key crypto operations thereafter (and possibly self signing a certificate if we follow part (b)'s approach), is an expensive crypto operation being incurred by S per connection. Hence the first approach is more efficient.

Problem 2. *(Stealth Port Scanning)*

Recall that the IP packet header contains a 16-bit identification field that is used for assembling packet fragments. IP mandates that the identification field be unique for each packet for a given (SourceIP,DestIP) pair. A common method for implementing the identification field is to maintain a single counter that is incremented by one for every packet sent. The current value of the counter is embedded in each outgoing packet. Since this counter is used for all connections to the host we say that the host implements a global identification field.

a) Suppose a host P (whom we'll call the Patsy for reasons that will become clear later) implements a global identification field. Suppose further that P responds to ICMP ping requests. You control some other host A. How can you test if P sent a packet to anyone (other than A) within a certain one minute window? You are allowed to send your own packets to P.

b) Your goal now is to test whether a victim host V is running a server that accepts connections to port n (that is, test if V is listening on port n). You wish to hide the identity of your machine A and therefore A cannot directly send a packet to V, unless that packet contains a spoofed source IP address. Explain how to use the patsy host P to test if V accepts connections to port n.
Hint: Recall the following facts about TCP:
– A host that receives a SYN packet to an open port n sends back a SYN/ACK response to the source IP.
– A host that receives a SYN packet to a closed port n sends back a RST packet to the source IP. A host that receives a SYN/ACK packet that it is not expecting sends back a RST packet to the source IP.
– A host that receives a RST packet sends back no response.

Problem 3. *(WPAD)*

WPAD is a protocol used by IE to automatically configure the browser's HTTP and HTTPS proxy settings. Before fetching its first page, IE will use DNS to locate a WPAD file, and if one is found, will use its contents to configure IE's proxy settings. If the network name for a computer is pc.cs.stanford.edu the WPAD protocol iteratively looks for wpad files at the following locations:

```
http://wpad.cs.stanford.edu/wpad.dat
http://wpad.stanford.edu/wpad.dat
http://wpad.edu/wpad.dat     (prior to 2005)
```

a) Explain what capabilities were inadvertently given to the owner of the domain wpad.edu as a result of this protocol. Explain how personal information can be exposed as a result of this issue.

b) Are pages served over SSL protected from the problem you described? If so, explain why; if not, explain why not.

Problem 4. *(CSRF Defenses)*

a) In class we discussed Cross Site Request Forgery (CSRF) attacks against web sites that rely solely on cookies for session management. Briefly explain a CSRF attack on such a site

b) A common CSRF defense places a token in the DOM of every page (e.g. as a hidden form element) in addition to the cookie. An HTTP request is accepted by the server only if it contains both a valid HTTP cookie header and a valid token in the POST parameters. Why does this prevent the attack from part (a)?

c) One approach to choosing a CSRF token is to choose one at random. Suppose a web site chooses the token as a fresh random string for every HTTP response. The server checks that this random string is present in the next HTTP request for that session. Does this prevent CSRF attacks? If so, explain why. If not, describe an attack.

d) Another approach is to choose the token as a fixed random string chosen by the server. That is, the same random string is used as the CSRF token in all HTTP responses from the server over a given time period. You may assume that the time period is chosen carefully. Does this prevent CSRF attacks? If so, explain why. If not, describe an attack.

e) Why is the Same-Origin Policy important for the cookie-plus-token defense?

Problem 5. *(Firewalls and Fragmentation)*

The IP protocol supports fragmentation, allowing a packet to be broken into smaller fragments as needed and re-assembled when it reaches the destination. When a packet is fragmented it is assigned a 16-bit packet ID and then each fragment is identified by its offset within the original packet. The fragments travel to the destination as separate packets. At the destination they are grouped by their packet ID and assembled into a complete packet using the packet offset of each fragment. Every fragment contains a one bit field called "more fragments" which is set to true if this is an intermediate fragment and set to false if this is the last fragment in the packet. We discussed the way small offsets can allow the data in one fragment to overlap information in the preceding fragment. However, overlapping fragments should not occur in normal network traffic.

In class we mentioned that when fragments with overlapping segments are re-assembled at the destination, the results can vary from OS to OS. Give an example where this can cause a problem for a network-based packet filtering engine that blocks packets containing certain keywords. How should a filtering engine handle overlapping fragments to ensure that its filtering policy is not violated?

Problem 6. *(DNSSEC)*

DNSSEC (DNS Security Extensions) is designed to prevent network attacks such as DNS record spoofing and cache poisoning. Generally, the DNSSEC server for example.com will posses the IP address of www.example.com. When queried about this record that it possesses, the DNSSEC server will return its answer with an associated signature. If the DNSEC server is queried about a host that does not exist, such as doesnotexist.example.com, the server uses NSEC or NSEC3 to show that the DNS server does not have an answer to the query.

Suppose a user R (a resolver, in DNS terminology) queries a DNSSEC server S, but all of the network traffic between R and S is visible to a network attacker N. The attacker N may read requests from R to S and may send packets to R that appear to originate from S.

a) Why is authenticated denial of existence necessary? To answer this question, assume that S sends the same unsigned DOES-NOT-EXIST response to any query for which it has no matching record. Describe a possible attack.

b) Assume now that S cryptographically signs its DOES-NOT-EXIST response, but the response does not say what query it is a response to. How is an attack still possible?

c) A DNSSEC server may send a signed NSEC response to a query that does not have a matching record (such as doesnotexist.example.com). An NSEC response contains two names, corresponding to the existent record on the server that immediately precedes the query (in lexicographic order), and the existent record that immediately follows the query. For example, if a DNSSEC server has records for a.example.com, b.example.com, and c.example.com, the NSEC response to a query for (non-existent) abc.example.com contains a.example.com and b.example.com because these come just before and just after the requested name. To be complete, NSEC records also wrap-around, so a query for a non-existent name after the last existent name will receive an NSEC containing the last and first existent names.

How should the resolver use the information contained in NSEC records to prevent the attacks you described in previous parts of this problem?

d) NSEC leaks information that may be useful to attackers on the Internet. Describe how an attacker can use NSEC to enumerate all of the hosts sharing a common domain-name suffix. How is this information useful for attackers?

e) NSEC3 is designed to prevent DNS responses from revealing unnecessary information. NSEC3 uses the lexicographic order of hashed records, instead of their unhashed order. In response to a query without a matching record, NSEC3 will return the hashed names that are just before and just after the hash of the query. For example, on a server containing a.example.com, b.example.com, and c.example.com, if a hashes to 30, b to 10, c to 20, and abc to 15, the NSEC3 response to a query for abc.example.com would contain 10.example.com and 20.example.com. Hashed names are also assumed to wrap around, in the same way as unhashed names in NSEC.

Explain how a resolver should verify the validity of a response under NSEC3?