

Homework 4

Due: April 7, 2009, 11:59 PM EST

Please hand in your solutions as a pdf document

Problem 1 As we saw in class, WEP (Wired Equivalent Privacy) is a deprecated legacy standard for private wireless networking.

1. Describe how messages are encrypted in WEP. Why is it important to use different session keys each time? Describe an attack on a protocol similar to WEP; except this protocol uses the same secret key across sessions.
2. Suppose an adversary wants to alter the content of a message M sent using WEP. Specifically, suppose M 's last 4 bytes are the IP of the sender of the message, which is known to the attacker to be 25.112.12.34. Show how the attacker can persuade the access point that M was sent by a host with IP address 45.145.56.1.
3. What is the purpose of an *IP redirection* attack and how does it work? Apart from changing the destination address, what else needs to be taken into consideration by the attacker in order for him to succeed?
4. Does WEP provide client authentication? How does this work? How can it be attacked?

Problem 2 One method to protect the privacy of the sender and recipient of a message, while also providing protection for message content, is *onion routing*. This method is based on the following approach:

- Messages travel from source to destination via a sequence of proxies (“onion routers”) along a randomly-selected path.
- The last router in the path establishes a connection with the intended recipient.
- To prevent eavesdropping attacks, messages are encrypted between routers.

The “onion” metaphor illustrates the essence of the method: as each router receives the message, it “peels” a layer off of the packet by decrypting it with its private key, thus revealing the routing instructions meant for that router. Due to this arrangement, the data in an onion packet can only be revealed if it is transmitted to every router in the path in the order specified by the layering.

1. What does an intermediate node need to know in order to complete its step in the onion routing protocol?
2. What does an exit node need to know in order to function effectively?
3. Recently, the Tor onion routing system received negative publicity because an exit node was compromised, exposing user traffic. What can a user do to prevent this problem?

Problem 3 In *broadcast encryption*, the *broadcast center* encrypts media content (songs, movies, etc.) to be broadcasted to a group of *subscribers*. A subscriber holds a set of secret keys that are given by the broadcast center to the subscriber at registration time. The secret keys are used to decrypt the broadcast content. Periodically, the keys of invalid subscribers (whose subscriptions terminate) are revoked. An invalid subscriber should not be able to decrypt content broadcasted after the revocation of the subscriber.

There are several approaches to key management for broadcast encryption. A first approach is to let every subscriber share the same secret key k . The broadcast center uses a symmetric encryption scheme to encrypt the contents with key k . The encryption is efficient, as the broadcast center just encrypts the content once. However revocation is inefficient, because it requires transmitting the new key to each of the valid subscribers over a separate secure channel. Another approach is that each subscriber has a different key. The broadcast center encrypts the contents multiple times, once per valid subscriber, which makes encryption inefficient. On the other hand, the revocation of an invalid subscriber is easy in this case — the center stops encrypting content with the revoked subscriber’s key.

Your task is to design an efficient key-management scheme for broadcast encryption that is based on symmetric-key encryption and supports the revocation of subscribers. The scheme should satisfy the following properties, where n denotes the total number of subscribers.

1. Each subscriber stores $O(\log n)$ secret keys.
2. Broadcast content is encrypted once per session, with a single session key.
3. The revocation of a single subscriber is performed with $O(\log n)$ messages.
4. No separate secure channels are used for revocation.

Describe your scheme and analyze its efficiency and security properties. In particular, describe (*i*) what are the secret keys of a subscriber; (*ii*) how a broadcast is encrypted; (*iii*) how an encrypted broadcast is decrypted; (*iv*) how a subscriber is revoked; and (*v*) why a revoked subscriber cannot decrypt content broadcasted after the revocation time. Assume that the broadcast center starts with n valid subscribers. *Hint*: recall the binary tree for key management in the DRM lecture.

Problem 4 OpenSSL is the standard cryptographic library used by `ssh-keygen` to generate public and private key pairs. During the summer of 2008, some astute hackers discovered that the random seed used to generate keys was the process ID of the key-generating process, obtained by invoking `getpid()`.

1. Explain why this implementation is a major vulnerability.
2. Identify the public key we gave you, find its size, type, and fingerprint. Execute `man ssh-keygen` to find the arguments to do that.
3. Determine the private key for the public key we gave you. Use a VM and shared folders from one of the previous assignments.

We have provided you with the following support code in `/course/cs166/asgn/hw4`:

- `openssl.tar`: snapshot of a system with the weak OpenSSL. Extract it onto the VM where you have root, it is necessary to be root because you will be extracting a fake root environment.

- `initializevm`: this script will install the OpenSSL exploit on a machine if you pass as the argument the path to where you extracted `openssl.tar`.
- `Makefile`: makes a shared library out of the code in `getpid.c` and compiles `test.c`
- `getpid.c`: contains the function `getpid()` which overrides
- `test.c`: a tester program to make sure `getpid()` is overridden
- `exploit`: runs `make all` and then sets `LD_PRELOAD` to the shared library generated above, anything run in this script will have its `getpid()` overridden with the function you defined in `getpid.c`.
- `takey.pub`: the public key

Environment variable `LD_PRELOAD` allows you to link your own libraries with any code that you run. In this case we are going to use it to override the C function `getpid()`, a form of DLL injection. This is all taken care of for you with the exploit file. What it does is ensure that `ssh-keygen` gets whatever PID you return in your `getpid()` function, so you can control which key is produced. You can use whatever language you want to do this. Generate the private key for the public key we have provided. Make sure it has the same fingerprint and is therefore the one and only match! Hand in your source code along with your solution.