

Security

Books

The Code Book, by Simon Singh. Very interesting and readable history of cryptography. (also wrote *Fermat's Enigma*)

Web Security, Privacy, and Commerce, 2nd ed by Garfinkle and Spafford. Use and ramifications of security on the Internet.

Secrecy

Bytes are stolen/intercepted – prevent the interceptor from learning anything from the bytes

Bytes could be on a socket

Bytes could just be a file

Authenticity

Identify – someone is who they claim to be

An Internet connection – who is it?

A file – who authored it? Has it been altered?

Alice and Bob

Alice wants to send a message to Bob

Secrecy: Alice sends Bob a message. A bad guy, Carl, intercepting the message cannot read it.

Authenticity: Bob receives the message, and can tell that it really was written by Alice and has not been changed.

Security: Secret Key

Security depends on a secret “key” known only to the parties

If the key is well protected, the protocol can and should be public

Security Through Obscurity

Security through obscurity – not a good idea

e.g. a detail of the protocol is not publicized, but Alice and Bob know and depend on it

Kind of like, the key to the front door is under the mat.

Inevitably, “secret” parts of the protocol become known, and then the whole thing is blown.

Real security depends on secret keys, not an obscure method

Obscurity can be somewhat effective in a low-tech way – e.g. keep your passwords in a file named “resume.txt”.

Can be effective against robotic attacks that are set up only to deal with a “standard” installation of, say, a web server, but get confused if your directories are not laid out the regular way.

Tech: Symmetric Cryptography

Classical encryption with a “key”, or password

Key is a “shared secret” known only to the desired parties

The key is used for both encryption and decryption

vs. Interception (secrecy)

vs. Forged document (authentication)

Problem: key distribution – how do you arrange the shared secret in the first place? Historically, this is a huge problem – e.g. the German Enigma encryption machine and its daily keys in WWII

Application: Simple Cryptography

Terms

p = plain text (readable)

crypt() = encryption/decryption function

c = cipher text (garbled)

k = key (secret, essentially, a password)

Alice sends message to Bob

Alice computes $c = \text{crypt}(p, k)$;

Alice sends c over Internet, store in file etc. to Bob

Bob computes $p = \text{crypt}(c, k)$

Bob gets the message, and we have both security and authenticity

Works because only Alice and Bob know the shared secret k

Application: Password Login

A wants to log in to their account on B

A and B both know the shared secret – the password

A sends the password to B, to prove A's identity – authenticity

Problem: password can be intercepted – “sniffing” – this used to be the most common source of problems on campus when “password in the clear” protocols were in wide use.

Solution1: challenge-response scheme

Don't send the password itself, the A just needs to prove that they have the password. B sends random R to A. A computes hash(R+password) and returns to B, proving that client knows the password.

Solution2: create a secure channel between the A and B, then do the password dialog (see below)

Many old protocols (telnet, FTP), send the password in the clear. There are newer variants that avoid this problem.

Tech: “Public Key” Cryptography

One of the most amazing developments in applied mathematics in recent memory. Instead of one, secret key, there are two keys which go together as a pair: one “public” and one “private”

Diffie and Hellman at Stanford sketched the original idea. Rivest, Shamir, Alderman (RSA) came up with the mechanics (patented until recently).

Property #1: A message can be encoded with one key and decoded with the other. Either key may be used to encode, and the other key decodes. Aka – “asymmetric” cryptography

Property #2: someone can know the public key, but knowing that will not allow them to deduce/compute what the private key is. As a practical matter, the scheme arranges that computing priv from pub requires solving a known difficult problem, such as factoring a huge number. If someone figured out a way to do factoring efficiently, then Property 2 would break.

Application: Public Key Secrecy

Bob generates a random pub/priv key pair on his own. Bob publishes the pub key.

Secure communications works as before, but the public key can be published. Alice can send secure mail to someone without first arranging a secret key – Alice just looks up Bob's public key knowing that Bob's private key will decode the message when he gets it. Like secret key encryption, but without the hassle of key distribution.

Alice looks up Bob's pub key...

Alice computes $c = \text{crypt}(p, \text{pub})$ and sends c to Bob

Bob computes $p = \text{crypt}(c, \text{priv})$ and reads the message.

The asymmetric property – the pub and priv keys go together – a message encoded with pub can only be decoded with priv (or the other way around), and only Bob knows what priv is. In fact, no one but Bob ever knew priv.

Tech: Hash Function

A 1-way hash function, that computes a short “digest” or “hash” of some bytes.

$h = \text{hash}(\text{input})$

1. Every bit in the input affects every bit in the output – changing one bit in the input results in a completely different output.
2. The function is not inevitable – given h , it is difficult (or believed to be difficult) to compute what the input was

Application: Error Detection

Suppose you are sending a large message.

Compute a “digest” $h = \text{hash}(\text{message})$ and include h at the end of the message.

The recipient can check that the message and the digest still match.

Application: Digital Signatures

Bob has a document, doc

Bob computes $h = \text{hash}(\text{doc})$

Bob uses his private key, to encode the digest resulting in a signature:

$\text{sig} = \text{crypt}(h, \text{priv})$

Bob attaches the signature to the document

Alice can look at the $\text{doc}+\text{sig}$, and compute that it really comes from Bob and goes with this document...

Alice computes $h = \text{crypt}(\text{sig}, \text{pub})$, and checks that $h = \text{hash}(\text{doc})$

Only Bob has priv, and so only Bob could have computed sig – the document must come from Bob. For this to work, Alice needs some independent way to look up Bob's public key – this is what a Certificate Authority (CA) does.

Note that this is better than a (ink, paper) signature – you cannot lift a signature from one document and put it on another. Bob's signature is tied to the particular document. Alice can verify Bob's signature, but Alice cannot pretend to be Bob.

Technology: Certificates

A Certificate Authority (CA) publishes that a particular identity (a person, an email address, a DNS name like store.foo.com), goes with a particular public key.

The CA does not know the private key.

The CA provides a certificate that associates the identity with the public key. The certificate is signed by the CA, so you know it is not just made up (if you trust the CA).

You can tell your browser which CA's to trust, and you can add them. The most common ones are built in.

CA's are a bit expensive and somewhat obnoxious at present, since there is a lack of competition – Verisign and Thawte are the largest

Application: HTTPS/SSL

This is how “secure” web connections/ ssh terminal connections work

Alice connects to Bob's server

Bob's server returns a certificate, and a token encrypted with Bob's priv key

Alice can verify that the certificate is valid, and use the public key to decrypt the token. If the token decrypts cleanly, Alice verifies that the other party really is Bob.

Alice can make up a one-time session secret, k , encrypt it under Bob's pub key, and send it to Bob. Only Bob will be able to recover the session secret → now the 2 have a shared secret, k , for simple cryptography for the session.

Attacks

CERT

Computer Emergency Response Team

Organization out of CMU that tracks vulnerabilities

<http://www.cert.org/>

<http://www.cert.org/advisories/>

There's some tension about when vendors should be told vs. the general public.

Vendors tend to like to keep it all quiet. However, it's bad if the bad guys know about the problems and the ordinary people do not

Also, the public humiliation of the vulnerability prompts the vendor to actually do something far more than a few customers complaining.

Typical practice is that the vulnerability will be kept quiet for a few weeks, until the vendor has a patch ready to go.

Attack Categories

Social engineering

Sniffing traffic

Spoof, inject traffic

Trojan horse

Virus, worm

Attack: Social Engineering

Tricking people socially into moving a file, revealing a password, etc.

This works very well – send the message “your password has been compromised, please change it to ‘n0wsafe’” – send this message to 10,000 AOL users – you get some.

Call the corporate help desk and pose as a confused user, lost password, etc.

By most accounts, social engineering can be very effective.

Attack: Traffic Sniffing

Looking at packets

Requires access to a machine on the IP route

Observe passwords or other info in the packets

At one time, this was the largest source of problems on campus. However “password in the clear” protocols are more rarely used, so now Windows/email/IIS vulnerabilities are now the largest problem.

Ethernet "promiscuous" mode

Most Ethernet hardware can be set in "promiscuous" mode where it logs all packets on the ethernet segment.

Script Kiddie / Rootkit

There are pre-made scripts on the net to aid attacks

The bad guys are called “script kiddies” to mock their lack of basic technical skill and maturity (they are often teenagers, using tools they don't really understand)

A “rootkit” used by a bad guy on a machine may...

1. Turn on promiscuous mode to try to capture passwords on the local ethernet
2. Replace the local network utility program with a version that reports that promiscuous mode is off.

Once a machine has been compromised, it's hard to fix. The rootkit may have replaced, say, ls so with a version that just does not show the bad guy files.

3l337 – Elite Speak

AKA 'leet speak (l33t)

Substitute digits for letters, and generally hash things up.

Own – get root access on a machine

31ee7 d00dz

A way of avoiding simple text search

Attack: Replay, Spoof

Carl observes Alice sending encrypted packets to Bob

Carl cannot read the packets, but he can re-send them later

Problem: Bob gets the “send this order to Alice” 10 times

Solution: include serial numbers, the current time, etc. in each message

Packet spoofing – bad guy inserts their own packets as if they were part of the stream of packets (difficult)

Solutions: certificates, encryption

Attack: Password Guessing

Many passwords are just a word, name, date,

Just use brute force

We may need personal ID gadgets someday, as human capacity to remember passwords hits its limit

Bad Guess => Deactivate, used as a DOS

Server could turn off an account if too many bad passwords

Can be used as a Denial of Service (DOS) attack (below) against someone

e.g. Disable the ebay account of a rival bidder

DOS

Attack: Denial Of Service (DOS)

Make a service unavailable

e.g. SYN flood – send many SYN packets (to start 3-way handshake) but never complete the handshake to overwhelm the victim

DOS attacks often involve malformed IP packets

Bad guy has robots launch a DOS attack on a server. High volume of traffic, hard to screen, and robots help shield identity of bad guy

Solution: partial solution is to firm up the internal routing protocols so it is harder to send bad packets – screen them out before they get to the victim.

Distributed DOS (DDOS)

Distributed DOS – bad guy takes over many machines for use as “robots.” e.g. a Win XP machine on a cable modem

On the Bad guy's signal, the hundreds of robots send traffic to the victim – greater effect, and it's harder to trace back to the bad guy

This technique works well – countless Windows machines out there have been taken over. Mostly they function fine and the owner has no idea about what is going on.

Executable Content

Code From Outside = Dangerous

Code is fundamentally dangerous compared to, say, a passive text document.

If Carl can execute his code on Alice's machine, Carl, in some sense, has control or use of Alice's machine

Code Solutions

Solution 1: don't use executable content – send text, images, pdf, ... content that intrinsically lacks execution.

Solution 2: code signing – the code is signed, so you know who it's from (Microsoft is working on this)

Solution 3: sandbox – the code runs in a limited environment where it can't cause much damage (Java does this and also code-signing)

Executable Content Vulnerability

A program, such as Internet Explorer, either by design or due to a bug, executes content sent to it. If the bad guy can execute code on a machine, they can use it for their own purposes

This has been a huge source of Microsoft vulnerabilities, as Microsoft frequently uses executable VB script embedded in documents as a way to integrate programs, not being fully aware of the security implications. This is a great way to get applications to work with each other (Word, Excel, ...), but it's a huge security problem.

Buffer Overflow Vulnerability

Suppose an FTP server is running on Alice's machine

Bob sends an over-sized request to the FTP server, exploiting a bug in the FTP server code that causes the FTP server to come under Bob's control (running on Alice's machine).

There have been many such vulnerabilities – but they are gradually being closed as programmers learn about that sort of bug.

Code

```
char* input = ...;
char buff[1000];
...
sscanf(input, "...%s...", buff);
```

Worm sends large data, which goes past the end of buff and writes things on the stack, altering return addr or something – very tricky to get details right, depends on stack layout, CPU type, etc.

The End Of Buffer Overflows

I believe that the most common sorts of buffer overflow vulnerabilities will be dying out soon. There will always be room for subtle vulnerabilities, but the big obvious holes are going away.

1. Programmers increasingly understand it
2. The operating system can re-arrange the stack in such a way as to make such attacks much more difficult
3. Languages such as Java are naturally immune.

Attack: Nimda

Very successful worm, uses many strategies.

1. Attacks many known Microsoft IIS web server vulnerabilities
2. On entry, sends copy of itself as a readme.exe to email addrs from addr book and internet history. These can execute on the recipient machine due to a IE 5,6 vulnerability. From: headers may be forged.
3. Installs the readme.exe in the web server doc tree, disguised as a .wav, so that all visiting users using IE will be infected when it automatically runs when they see the page
4. Local network aware – installs the exploit on locally writeable network shares

All these exploits were known and patchable, but obviously not 100% of users can have 100% of patches installed at all times.

Other Recent Attacks

Code Red – like Nimda, but less sophisticated

Spread versions of worm used same random seed – therefore they all checked the same hosts – oops! Fixed in Code Red v. 2

I Love You, etc. – just use email, disguise executable content to take advantage of IE, Outlook vulnerabilities

Solution: tools need to not execute content without specific action, warning, etc..

Other solutions: code signing, sandboxes

"Warhol" Worm

"Warhol" worm – cover the whole Internet in 15 minutes

Use a deliberate strategy to spread from infected to not-yet-infected hosts, without too much duplication or repetition of effort

Nicholas Weaver paper – How to Own the Internet in your spare time

<http://www.cs.berkeley.edu/~nweaver/cdc.web/>

Blaster Worm

Exploits Win32 RPC vulnerability using TCP port 135. Downloads mblast.exe into system32 directory, and runs it. Also DOS's the Windows Update server, so you can't patch.