Proper Use of Cryptography

- Never write your own crypto functions if you have any choice
  - Another favorite piece of advice from industry
- Never, ever, design your own encryption algorithm
  - Unless that's your area of expertise
- Generally, rely on tried and true stuff
  - Both algorithms and implementations

Proper Use of Crypto

- Even with good crypto algorithms (and code), problems are possible
- Proper use of crypto is quite subtle
- Bugs possible in:
  - Choice of keys
  - Key management
  - Application of cryptographic ops

An Example

- An application where RSA was used to distribute a triple-DES key
- Seemed to work fine
- Someone noticed that part of the RSA key exchange was always the same
  - That's odd . . .

What Was Happening?

- Bad parameters were handed to the RSA encryption code
- It failed and returned an error
- Which wasn't checked for
  - Since it "couldn't fail"
- As a result, RSA encryption wasn't applied at all
- The session key was sent in plaintext . . .

Another Example

- Many pieces of malware use cryptography
- RC4 is a frequent choice of cipher
  - Seems easy and fast
- So the hackers implement it themselves
- Which often gives the defenders advantages
- Because the hackers screw it up
- Being evil doesn't necessarily make you smart

Trust Management

- Don’t trust anything you don’t need to
- Don’t trust other programs
- Don’t trust other components of your program
- Don’t trust users
- Don’t trust the data users provide you

Trust

- Some trust required to get most jobs done
- But determine how much you must trust the other
  - Don't trust things you can independently verify
- Limit the scope of your trust
  - Compartmentalization helps
- Be careful who you trust

Two Important Lessons

1. Many security problems arise because of unverified assumptions
   - You think someone is going to do something he actually isn't
2. Trusting someone doesn't just mean trusting their honesty
   - It means trusting their caution, too

Input Verification

- Never assume users followed any rules in providing you input
- They can provide you with anything
- Unless you check it, assume they've given you garbage
  - Or worse
- Just because the last input was good doesn't mean the next one will be

Treat Input as Hostile

- If it comes from outside your control and reasonable area of trust
- Probably even if it doesn't
- There may be code paths you haven't considered
- New code paths might be added
- Input might come from new sources

For Example

- Shopping cart exploits
- Web shopping carts sometimes handled as a cookie delivered to the user
- Some of these weren't encrypted
- So users could alter them
- The shopping cart cookie included the price of the goods . . .

What Was the Problem?

- The system trusted the shopping cart cookie when it was returned
  - When there was no reason to trust it
- Either encrypt the cookie
  - Making the input more trusted
  - Can you see any problem with this approach?
- Or scan the input before taking action on it
  - To find refrigerators being sold for 3 cents

Variable Synchronization

- Often, two or more program variables have related values
- Common example is a pointer to a buffer and a length variable
- Are the two variables always synchronized?
- If not, bad input can cause trouble

An Example

- From Apache web server
- `cdata` is a pointer to a buffer
- `len` is an integer containing the length of that buffer
- Programmer wanted to get rid of leading and trailing white spaces

The Problematic Code

```
while (apr_isspace(*cdata))
  ++cdata;
while (len-- >0 &&
  apr_isspace(cdata[len]))
  continue;
cdata[len+1] = '/0';
```

- `len` is not decremented when leading white spaces are removed
- So trailing white space removal can overwrite end of buffer with nulls
- May or may not be serious security problem, depending on what's stored in overwritten area