CNS
Lecture 14

Writing secure code

Crypto APIs

Secure applications

594 paper due 12/1/06

Assignment 10 forensics

In the news

- Remember we said to avoid cross-site scripting use HTMLEntities() in PHP … well today, buffer overflows in this fucntion
- In case you hadn't noticed, about 9 out of 10 emails today are spam, sent by botnets around the world
- GNU Radius format string vulnerability
- Acer Notebooks ActiveX arbitrary command execution

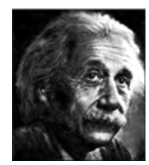
You are here …

Attacks & Defenses
- Risk assessment✓
- Viruses✓
- Unix security✓
- authentication✓
- Network security ✓
  Firewalls,vpn,IPsec,IDS
- Forensics ✓
- Secure coding

Cryptography
- Random numbers✓
- Hash functions✓
  MD5, SHA,RIPEMD
- Classical + stego✓
- Number theory✓
- Symmetric key✓
  DES, Rijndael, RC5
- Public key✓
  RSA, DSA, D-H,ECC

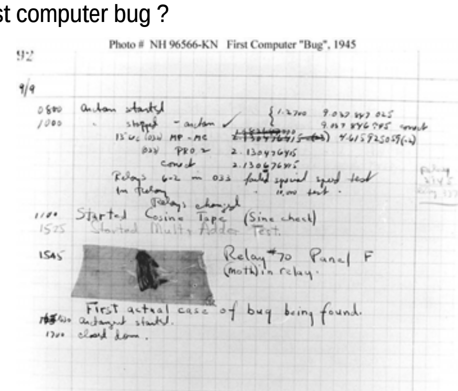
Applied crypto
- SSH ✓
- PGP ✓
- S/Mime ✓
- SSL ✓
- Kerberos ✓
- IPsec ✓
- Crypto APIs

Software security

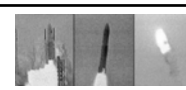


- Software engineering
  - Why do smart people write buggy software?
- Writing secure code
  - Secure design
  - Secure coding & tools
  - Testing for security
  - Defensive programming

First computer bug ?



Software bugs



- Flaws in software are expensive -- $60 billion/yr (NIST)
- Well-known bugs
  - Therac-25 (1985-1987) –radiation therapy machine killed 3 patients
  - Denver baggage handling system – complex (300 computers), late and 50% cost overrun, abandoned
  - Ariane 5 (1996) – loss of guidance system, flt. pt to 16-bit int overflow
  - USS Yorktown (1998) – aircraft carrier dead in the water for hours due to zero divide
  - Mars Climate Orbiter (September 23rd, 1999)-- $125m loss due to failure to convert meters to feet
- Just examples of when things go bad on their own. What if you have an active adversary who is looking for and exploiting bugs? Making an "unlikely event" a certainty in the millions of copies of the software in use around the world.

Bug density?

- 10 bugs per 1,000 lines of code?
  - Million lines of code → 10,000 bugs
- Complexity breeds bugs
- Of course all of these bugs aren't exploitable
- But it's a bug hunters paradise
  - The bad guys are winning

Lines of code (millions)	
NT 3.1	6
NT3.5	10
NT 4.	16
Windows 2000	29
Windows XP	40
Windows Vista	50
RedHat 7.1	30
Linux 6.0 kernel	6
Debian 2.2	56
Debian 3.1	213
Mac OS X 10.4	86
OpenSSL	0.2
Apache	0.2

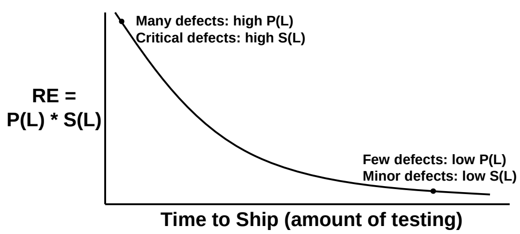
Why is software development so hard?

- Complex
  - No two parts the same
  - Growth adds to complexity in non-linear fashion
  - Interaction between data, function, user, devices
- Conformance to human requirements
- Changeable
  - Malleable
  - Easy to change, so let's change it
- Structure is impossible to visualize
- Human intensive
- Market pressures
  - Eliminate all bugs, never ship
  - Ship now, let customers debug

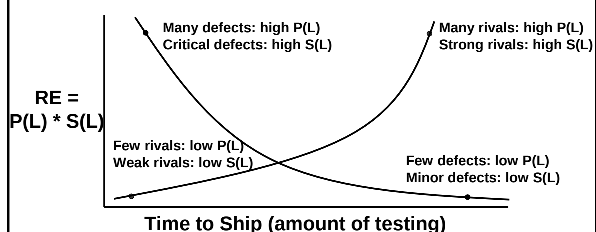
Risk Exposure

-Risk Exposure RE = Prob (Loss) * Size (Loss)
"Loss" – financial; reputation; future prospects, ...

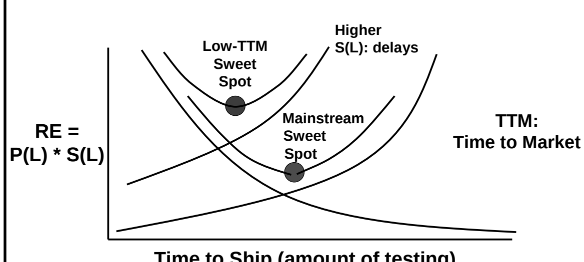
- Loss due to unacceptable dependability



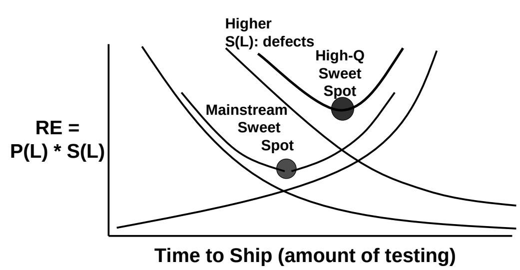
- Loss due to unacceptable dependability
- Loss due to market share erosion



Internet Startup

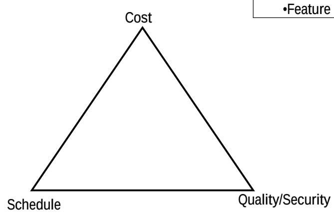


Safety-Critical or Secure System



Dilemma

Security conflicts with
- •Ease of use
- •Speed
- •Feature richness

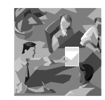


Pick any two.

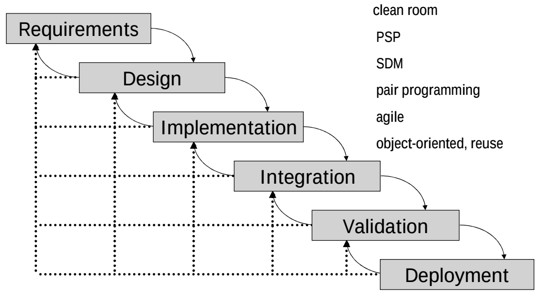
Software development

- You hack up some code
  - –Single developer
  - –“Toy” applications
  - –Short lifespan
  - –Single or few stakeholders
    - Architect = Developer = Manager = Tester = Customer = User
  - –One-of-a-kind systems
  - –Built from scratch
  - –Minimal maintenance

- Enterprise strength
  - –Teams of developers with multiple roles
  - –Complex systems
  - –Indefinite lifespan
  - –Numerous stakeholders
    - Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
  - –System families
  - –Reuse to amortize costs
  - –Maintenance accounts for over 60% of overall development costs



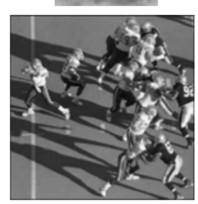
Software Development Lifecycle
Waterfall Model



Lots of methodologies
- clean room
- PSP
- SDM
- pair programming
- agile
- object-oriented, reuse

Playing the software development game …

- The trusting model
  - –Design/code/test to provide proper function
  - –Bugs are random and rare



- The defensive model
  - –Design/code knowing you have an active adversary
  - –Test not only that the right things happen, but also that wrong things don't happen
  - –Bugs may be vulnerabilities and exploited

Computers at Risk

“The developers of secure software cannot adopt the various probabilistic measure of quality that developers of other software can. For many applications, it is quite reasonable to tolerate a flaw that is rarely exposed and to assume that its having occurred once does not increase the likelihood that it will occur again. It is also reasonable to assume that logically independent failures will be statistically independent and not happen in concert. In contrast, a security vulnerability, once discovered, will be rapidly disseminated among a community of attackers and can be expected to be exploited on a regular basis until it is fixed.”

Software engineering for security

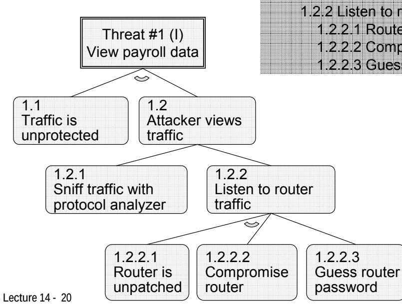
- Secure design
- Secure implementation
- Security testing
- Secure deployment
- Root cause analysis for bugs
- Policy and training

Secure design

- Assess the risk
  - Detail your assets
  - Know the threats and your attackers
  - Mitigate the threats
  - Costs: time & money … acceptable risk
- Formal threat modeling
  - Attack trees
  - Reduce the attack surface
- Apply secure design principles
- Think like a bad guy

Threat Modeling Process
Identify the Threats by Using Attack Trees

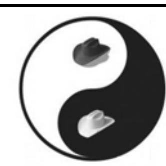
1.0 View payroll data (I)
    1.1 Traffic is unprotected (AND)
    1.2 Attacker views traffic
        1.2.1 Sniff traffic with protocol analyzer
        1.2.2 Listen to router traffic
            1.2.2.1 Router is unpatched (AND)
            1.2.2.2 Compromise router
            1.2.2.3 Guess router password



Design principles

- Principle of least privilege
  - Give only those privileges needed to complete a task
- Principle of fail-safe defaults
  - Access should be denied unless it is specifically permitted
- Principle of economy of mechanism
  - Security mechanisms should be as simple as possible
- Principle of complete mediation
  - All accesses to objects must be mediated
- Principle of open design
  - Security should not depend on secrecy of design or implementation
- Principle of separation of privilege
  - Don't grant permission based on a single condition (su: password+wheel grp)
- Principle of least common mechanism
  - Mechanisms used to access resources should not be shared
- Principle of psychological acceptability
  - Security mechanisms should not make resource access more difficult

How are the bad guys finding bugs?



- Open source ➔ code review
  - grep for str*()
  - Analyze input and packet parsing
- Blackbox testing
  - "random" alteration of input – big username, missing fields …
    - Cause a crash
    - Analyze dump
    - experiment
  - Disassembly (IDA Pro)
    - Look for str*(), network code
- Bug announcements, or diff patched vs old images
  - Attack unpatched versions
- Motivation?
  - Attribution
  - Financial gain

Secure implementation

- Trained developers
- Language choice – Java, C++, C#, C, VB, perl, php, …
- Trusted APIs (safestring, banned APIs)
- Development tools
  - Source code control
  - Compile/link/debug
  - Compiler warnings
- Peer code review
  - Specifically "security reviews"
  - Use checklists
- Static analysis
  - Microsoft PREfix and PREfast
  - RATS, ITS4,Klocwork, Coverity, ESC/Java

perl –T

Perl has option for reporting use of "tainted" variable at runtime

```
system("mail " . $form_data{"email"});
```


Code reviews

- Probably your best return on investment for security
- Will increase coding "cost", but worth it for production software
- Identify critical components for review
  - Processes that run with elevated privilege
  - Processes that talk to the net
  - Processes with "user input"
  - Software that parses packets/input
- Peer review source code and "patches" (person or team)
- Have a checklist of things to look for
  - Use of tainted data, banned functions, false assumptions
  - Failure to check return code, signed/unsigned, toctou
  - Update the list as bugs are found
- Sign off on review
- Engineers need to be trained to recognize flaws/vulnerabilities
- What about legacy code? Or outside libraries (e.g. OpenSSL)

Sample code

```
/* really bad telephone number lookup code, expects "name=foo" on stdin
static char cmd[128];
static char format[] = "grep %s phone.list\n";
main(){
  char buff[256];
  gets(buff);
  sprintf(cmd,format,buff+5);
  printf(cmd);  // debug, remove later
  system(cmd);
}


… more fragments
 p = malloc(sizeof(buff));
 strcpy(p,userdata);
 strncp(p,userdata,sizeof(p));   // better ?
 strncpy(p,userdata,sizeof(buff));   // better?
 buff[sizeof(buff)] = 0;


…
  char digest[16], tmp[16];
  …
 MD5_final(digest,&context)
 strncpy(tmp,digest,sizeof(digest));
```


More broken code snippets

```
char *p = NULL;
if (argc == 2)  p = argv[1];   //suggest if (2 == argc) … why
*p = 'A';


while(a=5){
   …


int fcn(){
  char *p;
  int i;
  p = malloc(1024);
  …
  return (i+5);
}
```

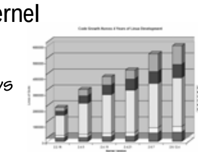

Static analysis

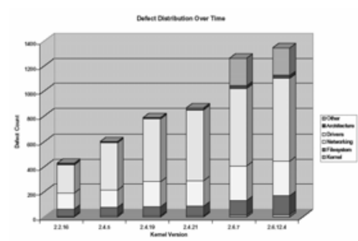
- Automated source code analysis
- Looking for
  - *Potential NULL pointer dereferences*
  - *Access beyond an allocated area (e.g. array or dynamically allocated buffer); otherwise known as a buffer overflow*
  - *Writes to potentially read-only memory*
  - *Reads of potentially uninitialized objects*
  - *Resource leaks (e.g. memory leaks and file descriptor leaks)*
  - *Use of memory that has already been deallocated*
  - *Out of scope memory usage (e.g. returning the address of an automatic variable from a subroutine)*
  - *Failure to set a return value from a subroutine*
  - *Buffer and array underflows*
  - if (x=3) …
- List of warnings, many false positives
- 10x slower than compile

General:	
Report file	apache.gor
Report date	Tue Aug 22 14:42:56 2006
Totals:	
Files	173
Functions	2008
Lines	93684
Errors and warnings	191
Errors	157
Warnings	34
Averages:	
Lines per function	46
Errors per file	0.908
Warnings per file	0.197
Errors per function	0.078
Warnings per function	0.017
Lines per error	596
Lines per warning	2755

Coverity's static analysis of Linux kernel

- *Growth in linux kernel in drivers*
- *Complexity grows but not % of security flaws*
  - *2001 2.4.1 kernel 1.6 MLOC 1000 flaws*
  - *2004 2.6.4 kernel 5 MLOC 900 flaws*





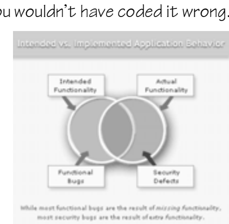
Coverity's analysis of open source software

Code Base	Lines of Code	Number of Errors	Analysis Time (min.)	Defect Density
Amanda	87,332	108	8	1.237
Apache	127,839	32	10	0.250
Ethereal	1,157,801	143	108	0.124
Firebird	238,701	163	13	0.680
Firefox	303,908	108	24	0.355
FreeBSD	1,587,166	635	257	0.401
Gaim	320,930	113	18	0.352
Gcc	692,860	140	65	0.202
Gnome	1,954,504	896	172	0.458
Ipecast	37,047	12	1	0.324
Jnetutils	71,892	29	4	0.403
Linux*	3,171,631	1062	254	0.335
Mplayer	484,554	284	38	0.586
MySQL	607,639	136	68	0.224
NetSNMP	173,138	148	16	0.855
OpenLDAP	254,304	158	20	0.622
OpenSSL	194,751	66	19	0.339
OpenVPN	69,610	7	4	0.101
Perl	479,758	89	25	0.186
PHP	430,817	204	36	0.474
PostgreSQL	815,562	295	38	0.362
ProFTPD	88,634	26	4	0.289
Python	258,272	96	16	0.372
Samba	310,592	216	34	0.695
Snort	82,919	48	4	0.579
SQLite	60,727	31	6	0.510
Squid	134,880	53	8	0.393
TCL	120,538	69	11	0.572
WxWidgets	303,283	73	39	0.241
X	2,353,980	1681	224	0.714
Xine	576,526	347	35	0.602
XMMS	116,768	6	4	0.051

- Less than 1 bug per 1,000 lines of code

Testing for security

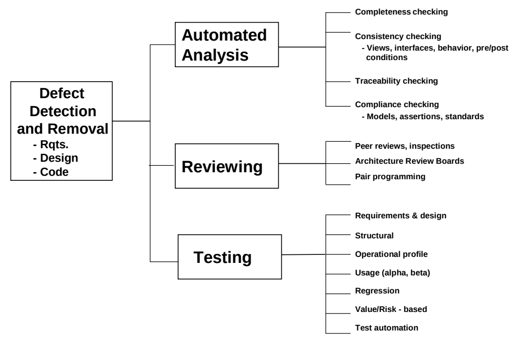
- Regression testing, unit testing, integration testing, system testing
- Most testing is aimed at testing conformance/quality, that code does what it's supposed to do
- Security testing is different, making sure bad things don't happen
  - –If you knew to test for it, you wouldn't have coded it wrong…
  - –Independent test group



Secure testing

- Run time tests
  - –Memory leak tester (purify)
  - –Fuzz testing, random user/packet input (udpsic)
  - –Smarter fuzz testing -- codenomicon
  - –Lots of web app testing products
- Penetration testing
  - –Think/attack like a bad guy
  - –Monitor bugtraq/CERT, because the bad guys are pen testing!

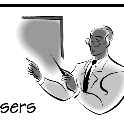
Software Defect Detection Opportunity Tree



Defense in Depth

- Training, code review, testing won't find all the bugs (prevention)
- Buffer overflow defenses
  - –Stackguard/propolice
  - –Hardware memory protection (RO/NX)
  - –Address randomization (text/heap/stack/data) … see PaX
- Fail gracefully, fault tolerance
- Log events and error conditions
- Watch for crashing daemons/servers

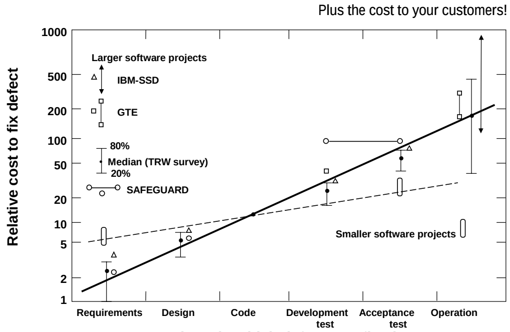
Security training



- Educate your designers, developers, testers, and users
- How can you do a risk assessment, if you don't know the risks?
- How can you do code reviews looking for security flaws, if you don't know what security flaws look like?
- Yearly re-train?
- Metrics? Bugs/1000 lines of code

Factor-of-100 Growth in Software Cost-to-Fix

Plus the cost to your customers!



Fixing bugs

- Root cause analysis
- Each defect found can trigger five analyses:
  - – Debugging: eliminating the defect
  - – Regression: ensuring that the fix doesn't create new defects
  - – Similarity: looking for similar defects elsewhere
  - – Insertion: catching future similar defects earlier
  - – Prevention: finding ways to avoid such defects
- Example, strcpy() cause for buffer overflows
  - – ban str*() and provide safe string libs
  - – Training
  - – Static analysis tools to detect

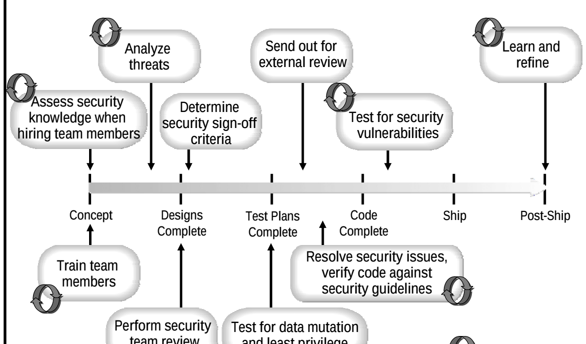
Software Defect Reduction Top-10 List
from CeBASE http://www.cebase.org

1. Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase.
2. About 40-50% of the effort on current software projects is spent on avoidable rework.
3. About 80% of the avoidable rework comes form 20% of the defects.
4. About 80% of the defects come from 20% of the modules and about half the modules are defect free.
5. About 90% of the downtime comes from at most 10% of the defects.
6. Peer reviews catch 60% of the defects.
7. Perspective-based reviews catch 35% more defects than non-directed reviews.
8. Disciplined personal practices can reduce defect introduction rates by up to 75%.
9. All other things being equal, it costs 50% more per source instruction to develop high-dependability software products than to develop low-dependability software products. However, the investment is more than worth it if significant operations and maintenance costs are involved.
10. About 40-50% of user programs have nontrivial defects.

Microsoft secure coding

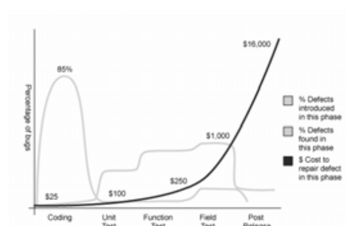
- Security push – audit legacy code for security flaws
- For Vista, Security Development Life Cycle
  Design – threat modeling, least privilege, reviews
  Implementation – banned APIs, tools, review, bluehat
  Deployment – secure by default, root cause analysis of bugs
- Security training – from one course to a dozen
- Address Space Layout Randomization (ASLR)
  - Randomize text/data/heap
  - Stackguard (/GS)
  - Support RO/NX
- TPMs and Bitlocker
- Upper management support

Secure Product Development Timeline



Design security in from the beginning

- Good design and code reviews are more effective than testing
- Should the software vendor be held accountable for loss due to software flaws?



Adding security in

- Our ncp labs (assignments 4, 7, 8)
  - Design problems
  - coding problems
- TCP protocol
  - Assumed "cooperating" processes
  - SYN flooding, malformed packets …
- NTP protocol
- SNMP protocol (Simple Network Management Protocol)
  - Or Security Not My Problem
  - Clear text read/write group (v1)
  - Added notion of views (ACLs/ least privilege) (v2c)
  - Added authenticity/privacy (v3)

SNMP

Simple Network Management Protocol
- Used to manage network devices (routers, switches, toasters,…)
- Network manager and data base (MIB) and network agents
  - ASN.1/BER data encoding
  - Object ID (OID) (long numeric tag) provides unique variable names (tree)
- Simple datagram protocol (UDP port 161)
  - GET GET-NEXT GET-RESPONSE SET TRAP (port 162)
- Version 1, simple (no) security (community string)
  - SET's disabled
  - Security for SNMP v1: Security Not My Problem
- Numerous implementation flaws discovered with Finland/Codenomicon smart fuzz tester
  - Buffer overflows community string
  - ASN.1/BER bugs (Tag Length Value TLV)

SNMP v3 security (RFC 3414)

Threats
- Impersonation
- Modification
- Replay/re-order/delay
- disclosure

Countermeasures (v3)
- Timeliness
  - Loosely synchronized monotonically increasing time indicators
- Authentication
  - MD5 HMAC (96 bit)
  - 128 bit (16-byte) key
  - SHA optional (crypto agile)
- Privacy
  - DES/CBC (AES & D-H option)
  - 8 byte key + 8 byte IV (from key)
  - Export restrictions ☹
- Additional control thru router filters

Key update:
1. administrator picks newkey
2. mgt. station generates random # rand and calculates delta = newkey ⊕ hash(rand, oldkey)
3. send client delta, rand
4. client recovers newkey = delta ⊕ hash(rand, oldkey)
   no Perfect Forward Secrecy ☹

Crypto APIs

- functions for developing crypto applications
- provide data privacy and integrity and authenticity
- portable -- architecture and OS
- services
  - establish context (keying, algorithm negotiation)
  - encrypt/decrypt and authenticate
  - certificate mgt. (fetch, CRL)
  - sign/verify
  - encode/decode (PKCS, ASN)
  - protocols (LDAP, SSL/TLS, key update)
- multiple algorithms -- symmetric, asymmetric, hash, random, primality, big numbers, compress, encode

Example CAPIs

- roll your own
  - build from algorithms/shareware
  - GNU's multiprecision lib
  - encoding/portability
  - Schneier PC disks
- shareware packages
  - OpenSSL or Young's libdes/SSLeay or RSA's BSAFE
  - Blaze's cryptolib
    - DSA, RSA, D-H, ElGamal, Rabin
    - DES/3DES, MD5, SHA
    - bigmath, truerand, primality, quantize
  - elliptic curve routines
- other: crypto++, JAVA crypto

Crytpo++

- Dai's C++ class library
- block ciphers (incl. AES)
- hashes/MACs (SHA/MD5/Tiger/RIPEM/Panama)
- RSA/DSA/ECC/D-H
- primes/PRNGs/compress/encode
- benchmarks
- class hierarchy with abstract base classes
- filter/pipeline metaphor

```
class MD5 : public IteratedHash<word32, false, 64>
{ ...  }

MD5 md;
SecByteBlock digest(md.DigestSize());

md.Update(testSet[i].input, testSet[i].inputLen);
md.Final(digest);
```

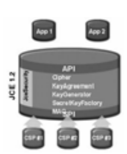
base classes

BlockTransformation
BufferedTransformation
BufferedTransformation::Err
Exception
FixedBlockSize
HashModule
PK_Signer::KeyTooShort
MessageAuthenticationCode
PK_AuthenticatedKeyAgreementDomain
PK_CryptoSystem
PK_Decryptor
PK_Encryptor
PK_FixedLengthCryptoSystem
PK_FixedLengthDecryptor
PK_FixedLengthEncryptor
PK_Precomputation
PK_SignatureSystem
PK_SignatureSystemWithRecovery
PK_Signer
PK_SignerWithRecovery
PK_SimpleKeyAgreementDomain
PK_Verifier
PK_VerifierWithRecovery
PK_WithPrecomputation
RandomAccessStreamCipher
RandomNumberGenerator
StreamCipher

Java JCE

- Exportable
  - Crippled crypto or signed applications
- Provider-based architecture (CSPs)
- hashes, block ciphers, PRNG
  - DES, AES, Blowfish, SHA, MD5, HMAC
- public key (RSA/DSA), SSL, and D-H
- certificate mgt (keytool)
  - public key generation
  - signing (CA)
  - signing JAR's
- also Crytpix API

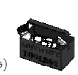
```
import java.io.*;
import java.net.*;
import java.security.*;
import javax.crypto.*;
...
KeyGenerator generator =
          KeyGenerator.getInstance("DES");
generator.init(new SecureRandom());
SecretKey key = generator.generateKey();
Cipher encipher =
Cipher.getInstance("DES/ECB/PKCS5Padding");
Cipher decipher =
Cipher.getInstance("DES/ECB/PKCS5Padding");
encipher.init(Cipher.ENCRYPT_MODE,key);
decipher.init(Cipher.DECRYPT_MODE,key);
...
ciphertxt = encipher.doFinal(cleartxt);
cleartxt = decipher.doFinal(ciphertxt);
.......................
Signature dsa =
Signature.getInstance("SHA1withDSA");
PrivateKey priv = pair.getPrivate();
dsa.initSign(priv);
dsa.update(data);
byte[] sig = dsa.sign();
```



Cryptographic Service Provider

OpenSSL API openssl.org

- widely used reference implementation
- Our basic C crypto toolkit
  - Hashing (MD*, SHA, RIPEMD)
  - Random numbers and prime numbers
  - Big number library
  - Symmetric key crypto (RC4, blowfish, AES, DES)
  - Public key crypto (RSA, DSA, D-H, ECC)
  - Support for crypto hardware accelerators (engines)
  - Crypto-agile wrappers (EVP)
- SSL/TLS
  - API for creating SSL network connection (asnmt 9)
  - Commands for key/certificate management (your own CA)
    - genrsa -- key generation
    - verify -- verify cert
    - pkcs12 -- convert cert encodings
    - x509 -- cert mgt
    - req -- generate cert or CSR
- Keep current
  - New features
  - Bug fixes



Example CAPI

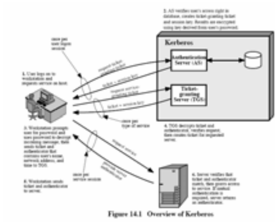


Figure 14.1 Overview of Kerberos

Kerberos

- Client context established (tickets, keys) KDC
- server/principal must be registered
- krb_mk_safe(), krb_mk_priv()
- user provides transport

DCE

- context established (tickets, keys) KDC
- server/principal must be registered
- rpc_binding_set_auth_info()
- RPC mechanism provides transport

Kerberos v4 API example

- client sends safe and private message
- server decodes and prints
- server must be registered with Kerberos and server key in server's /etc/srvtab (HOST and SERVICE in example C code)

```
krb_mk_req(authent,service,instance,realm,checksum)
krb_rd_req(authent,service,instance,from_addr,ad,fn )
krb_get_cred(service,instance,realm,credentials)
krb_mk_priv(in,out,in_length,schedule,key,sender,receiver)
krb_rd_priv(in,in_length,schedule,key,sender,receiver,msg_data)
krb_mk_safe(in,out,in_length,key,sender,receiver)
krb_rd_safe(in,length,key,sender,receiver,msg_data)
```


Kerberos v4 client

```
main()
{
  KTEXT_ST k;        /* Kerberos data */
  KTEXT ktxt = &k;
  CREDENTIALS c;       /* ticket & session key */
  CREDENTIALS *cred = &c;
  des_key_schedule sched;    /* session key schedule */
  if ((serv = getservbyname(SERVICE, "udp")) == NULL) {
         fprintf(stderr, "service unknown: %s/udp\n", SERVICE);
         exit(1);
  }
  if ((host = gethostbyname(HOST)) == (struct hostent *) 0) {
         fprintf(stderr, "%s: unknown host\n", HOST);
         exit(1);
  }
  bzero((char *)&s_sock, sizeof(s_sock));
  bcopy(host->h_addr, (char *)&s_sock.sin_addr, host->h_length);
  s_sock.sin_family = AF_INET;
  s_sock.sin_port = serv->s_port;
  gethostname(hostname, sizeof(hostname);
  host = gethostbyname(hostname)
  if ((sock = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
         perror("opening datagram socket");
         exit(1);
  }
```


Client continued …

```
bzero((char *)&c_sock, sizeof(c_sock));
bcopy(host->h_addr, (char *)&c_sock.sin_addr, host->h_length);
c_sock.sin_family = AF_INET;
if (bind(sock, (struct sockaddr *)&c_sock, sizeof(c_sock)) < 0) {
 perror("binding datagram socket");
 exit(1);
}
/* Get local realm, not needed, just an example */
if ((i = krb_get_lrealm(c_realm, 1)) != KSUCCESS) {
 fprintf(stderr, "can't find local Kerberos realm\n");
 exit(1);
}
printf("Local Kerberos realm is %s\n", c_realm);
/* Get Kerberos realm of host */
s_realm = krb_realmofhost(HOST);

/* Get ticket for server from TGS, create krb_mk_req message */
if ((i = krb_mk_req(ktxt, SERVICE, HOST, s_realm, cksum))!= KSUCCESS) {
 fprintf(stderr, "%s\n", krb_err_txt[i]);
 exit(1);
}
printf("Got credentials for %s.\n", SERVICE);
```


Client cont.

```
  /* Send authentication info to server */
 i = sendto(sock, (char *)ktxt->dat, ktxt->length, flags,(struct sockaddr *)&s_sock,
 sizeof(s_sock));
 if (i < 0)        perror("sending datagram message");
 printf("Sent authentication data: %d bytes\n", i);
  /* PREPARE KRB_MK_SAFE MESSAGE */
  /* Get my address */
 bzero((char *) &c_sock, sizeof(c_sock));
 i = sizeof(c_sock);
 getsockname(sock, (struct sockaddr *)&c_sock, &i);
  /* Get session key from my collection of tickets */
 i = krb_get_cred(SERVICE, HOST, s_realm, cred);
 /* Make the safe message */
 len = krb_mk_safe(MSG, ktxt->dat, strlen(MSG)+1,cred->session, &c_sock, &s_sock);
  /* Send it */
 i = sendto(sock, (char *)ktxt->dat, (int) len, flags, (struct sockaddr *) &s_sock,
 sizeof(s_sock));
  /* PREPARE KRB_MK_PRIV MESSAGE */
  /* Get key schedule for session key */
 key_sched(cred->session, sched);
  /* Make the encrypted message */
 len = krb_mk_priv(MSG, ktxt->dat, strlen(MSG)+1,sched, cred->session, &c_sock, &s_sock);
  /* Send it */
 i = sendto(sock, (char *)ktxt->dat, (int) len, flags, (struct sockaddr *) &s_sock,
 sizeof(s_sock));
}
```


Kerberos v4 server

```
main()
{
  KTEXT_ST k;
  KTEXT ktxt = &k;     /* Kerberos data */
  AUTH_DAT ad;         /* authentication data */
  struct sockaddr_in c_sock; /* client's address */
  MSG_DAT msg_data;    /* received message */
  des_key_schedule sched;    /* session key schedule */
/* Set up server address */
  bzero((char *)&s_sock, sizeof(s_sock));
  s_sock.sin_family = AF_INET;
  serv = getservbyname(SERVICE, "udp");
  s_sock.sin_port = serv->s_port;
  gethostname(hostname, sizeof(hostname);
  host = gethostbyname(hostname);
  bcopy(host->h_addr, (char *)&s_sock.sin_addr, host->h_length);
  sock = socket(AF_INET, SOCK_DGRAM, 0);
  bind(sock, (struct sockaddr *)&s_sock, sizeof(s_sock));
/* GET KRB_MK_REQ MESSAGE */
  i = read(sock, (char *)ktxt->dat, MAX_KTXT_LEN);
  ktxt->length = i;
/* Check authentication info */
  i = krb_rd_req(ktxt, SERVICE, HOST, any, &ad, "");
  if (i != KSUCCESS) {
...
```

Receive ticket from client

Verify ticket and extract session key using server key in /etc/srvtab

Server cont.

```
/* GET KRB_MK_SAFE MESSAGE */
i = sizeof(c_sock);
i = recvfrom(sock, (char *)ktxt->dat, MAX_KTXT_LEN, flags,
   (struct sockaddr *)&c_sock, &i);
/* Verify the checksummed message */
i = krb_rd_safe(ktxt->dat, i, ad.session, &c_sock, &s_sock, &msg_data);
if (i != KSUCCESS) {
....

printf("Safe message is: %s\n", msg_data.app_data);
/* NOW GET ENCRYPTED MESSAGE */
key_sched(ad.session, sched);
i = sizeof(c_sock);
i = recvfrom(sock, (char *)ktxt->dat, MAX_KTXT_LEN, flags,
(struct sockaddr *)&c_sock, &i);
i = krb_rd_priv(ktxt->dat, i, sched, ad.session, &c_sock, &s_sock, &msg_data);
if (i != KSUCCESS) {
...
printf("Decrypted message is: %s\n", msg_data.app_data);
}
```

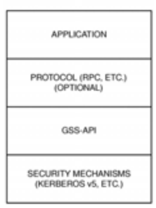

kerberizing

- you can add Kerberos calls to your own client/servers
- need Kerberos data base, authenticator, ticket-granting server, and administrative programs
- can use klogin, but better if you have kerberized BSD utilities
- Kerberos calls added to login, r-utilities, NFS

GSS

Generic Security Service

- IETF RFC 1508 1509, API: RFC 2743 2744
- establishes a security context and provides security services
- application goes through 4 phases
  1. establish identity (authentication)
  2. negotiate shared security context
  3. exchange messages -- privacy, integrity
  4. destroy context
- sample implementations in Kerberos v5 and DCE distributions
- API is independent of OS and network protocols
- application is responsible for message transport or file I/O
- used in globus (grid security)
- Java class GSSUtil
- primary functions
  - gss_acquire_cred()
  - gss_init_sec_context()
  - gss_seal() gss_unseal()
  - gss_sign() gss_verify()



Example CAPI

PKCS 11 cryptoki/ FORTEZZA

- crypto token/smartcard API
- routines for
  - –card discovery
  - –PIN verify
  - –encrypt/decrypt
  - –hash/sign/verify
  - –wrap/unwrap key
  - –random generation
  - –management functions

Example CAPI

•actual algorithms (RSA, DES, MD5) provided by cryptographic service provider (CSP) can replace underlying CSP.
•govt's or vendors might define CSP's (e.g., FORTEZZA).
•RSA CSP is provided.
•Microsoft must sign the CSP.
•supposedly easy to switch from strong to weak crypto by swapping CSP
•used in/since Internet Explorer 3.0
•samples and manual available

Microsoft CAPI

- 23 crypto services
- context and key generation
- key exchange
- encryption/signing

CryptAcquireContext() CryptGenKey() CryptGetKey()
CryptEncrypt() CryptDecrypt() CryptHashData()
CryptSignHash() CryptVerify() **CryptGenRandom()**

Other proprietary CAPIs – Spyrus or Cyrix

Where to encrypt?

link layer

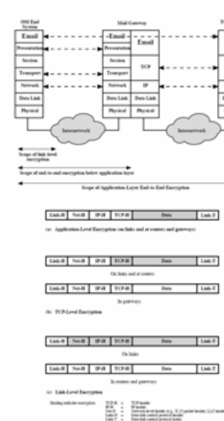
- encrypting modem, net board (wireless)
- transparent, fast
- protects only one link (pt-to-pt)
- info may be exposed in OS

network/transport layer

- IPsec IPv6(v4), VPN
- transparent
- selectable (policy)
- appl./host/net keying

application layer

- custom applications (PGP, ssh)
- CAPI's can help (openSSL)
- intrusive, but flexible
- key for every logical circuit



Secure applications

App's we've already seen

- ssh
- PGP
- SSL/netscape
- S/MIME
- characteristics
  - key mgt/schedule
  - ciphers/hashes
  - public key/symmetric key
  - random numbers
  - Protocols and encodings
  - Crypto agile

nautilus

nautilus sinks clipper ships ...

- encrypted Internet/modem phone
- UNIX, Windows
- various audio encodings
- D-H or shared secret
- 3DES, Blowfish, IDEA
- random bits from microphone noise

speakfreely

- Use microphone/speaker for secure voice over IP
- Uses GSM compression, SGI's AtoD PCM, VAT/RTP options
- Crypto
  - AES/Blowfish/IDEA/DES to encrypt audio (CBC within a "block")
  - Session key can be provided by user as ASCII passphrase or by a key file (e.g. dd count=10 of=rand.dat if=/dev/random)
  - sfmike –e will generate a session key (128 bit encoded into 32 ASCII characters that Bob could PGP email to Alice, or read over phone ?)
  - For "conference call" -Z option will create session keys for each user and PGP/GPG encrypt the key with their public key!

```
makeSessionKey() in mike .c creates random session key wtih
     getpid() getppid()  clock() time() gethostid() gethostname()
     getuid() geteuid() getdomainame()  and  /dev/random
 all hashed with MD5
```


zfone

- Zimmerman's voice-over-IP secure phone (IETF draft)
- Uses ideas from PGPfone
- Doesn't need PKI
  - Does peer-to-peer key establishment
- Header extensions to RTP to support Diffie-Hellman
  - Establishes ephemeral session key/IV for Secure RTP
  - Displays authentication string for each user to verify authenticity and prevent man-in-the-middle attacks
  - Can include pre-shared secret and running shared secret
- Secure RTP uses
  - HMAC-SHA1
  - AES 128 (counter mode)
  - 112-bit session salt key
  - 2^{48} key derivation rate

Disk encryption cfs

```
startup cfs
cfsd
mount -o port=3049,intr localhost:/null /crypt

Create encrypted directory
cmkdir /usr/dunigan/secrets
Key: xxxxxxxxxxxxx

cattach /usr/dunigan/secrets tom
Key: xxxxxxxxx      (now have /crypt/tom)

do normal file operations cat, vi, etc.

cdetach tom

/usr/dunigan/secrets/80b6e856778
contents are gibberish (encrypted)
```

- des or pgp -c is cumbersome
- Blaze AT&T
- most BSD-based UNIX and linux
- local NFS server on loopback interface
- encrypted directories
- backups etc. are unaffected
- DES/3DES/MacGuffin
- /crypt is mounted
- command to attach/detach (with DES key)
- if CFS unavailable, ccat can be used

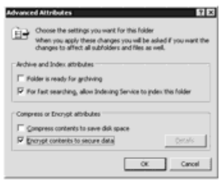
cfs encryption

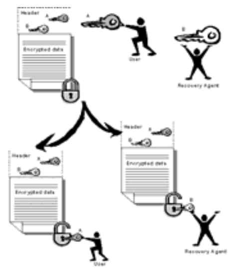
- need random access for file
  - ECB works but crypto weak
  - CBC can do random reads, but need to re-encrypt all to do block update
- cfs computes two DES keys from passphrase
- uses first key to create 512KB pseudo-random bit mask using DES-OFB -- stored for life of cattach
- file block is XOR'd with mask corresponding to byte offset in file mod size of mask (512KB) and then encrypted with DES-ECB
- optionally file IV's generated from inode number and stored in file's group-id field
- 4 times slower than regular NFS I/O benchmarks, but only about 1.3 slower for "normal" applications (compile)

Are plaintext versions lying around ? swap space, backup files (vi), /dev/mem ...
wipe utility to over-write plaintext?
Is the passphrase lying in memory/swap space ?
Unattended terminal/session? -- timeout?

Windows Encrypting Files System (EFS)

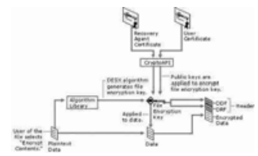
- Encrypt folders (DES-X or 3DES)
- Provides key recovery, you need a certificate



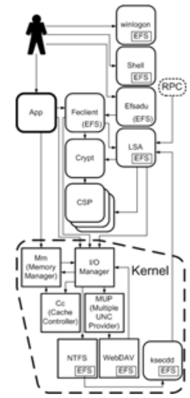


Windows EFS

Windows generates a random number for the file encrypting key (FEK). User's public key is used to encrypt the FEK. Encrypted key is stored in extension attribute to the file. FEK also encrypted under recovery agent's public key.

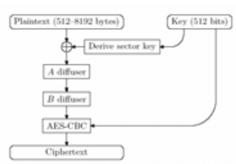


"wipe" service provided to scrub clear text file blocks.



Bitlocker vista

- Drive encryption optionally using TPM, optional user password
- Solve the stolen laptop problem
  - Removing the drive or booting with thumb drive and becoming Administrator still doesn't give you access to the bitlocked disk
- Each sector encrypted with AES CBC
  - no authentication ? Because it would require more bits per sector
  - Since CBC any change will effect the rest of the sector
  - Uses sector number as part of encryption
  - IV is encryption of sector key and sector number
- Uses "Elephant diffuser"
  - Provide poor man's authentication
  - Fast (32-bit word XOR and rotate)
- Performance (pentium 4)
  - AES 20 cycles/byte
  - Diffuser 10 cycles/byte
  - 5% slower



MAC disk encryption

- Encrypt a disk "volume" using AES
  - Key can be added to your keychain for one-time signon
- FileVault to AES-encrypt your home directory
  - Key recovery option with Master password
  - Disable auto-login (duhhh)
  - Backups unencrypted (unless you backup the vault)

Next time

Review