Distributed System Security: The challenges and big picture

15-712
David Andersen

What is the target?

- Whatever the weakest link in the system is!
- "The System" means *everything and everyone* involved
  - e.g., ATM example:
    - Programmers, managers, mailers, postal system, users, tellers, algorithms, cryptography, implementation details
      - Programmers: crooks, bribe/blackmail, bugs
      - Employees: sell acct #s and PINs; address change example
      - Users: social engineering attacks ("hi, this is Dave from Credit Management Union, could I get your PIN?")
      - Crypto: oops, we used a Caesar cipher (WiFi?)
      - impl bugs: c.f. worm paper, buffer overflows, misunderstanding of security properties and layer interactions, ...
    - Human factors and technical factors
      - Humans are often easier to trick

Modern analogs

- ATMs are still a popular thing to attack
  - Latest: Fake readers, POS terminals, cameras
    - Smartcards a possible solution, but $
- Worms:
  - Some spread through exploits (morris, code red, etc.)
  - Many spread through user error ("click me!") (Storm)
    - Biggest bot-nets in the world are human exploits
- Phishing
- TJX breakin (software vuln)
- SQL injection attacks (programmer errors--we haven't made it easy enough to do the right thing.)

Real world is messy

- Multiple vulnerabilities *common*
  - e.g., exploit social channel to get user account; use local privilege elevation to get root -- very common in targeted attacks.
  - Morris worm: get local accounts, hack passwords, exploit buffer overflows, exploit trust relationships, ...

Passwords

- UNIX /etc/passwd file
  - Stores crypt()ed version of password
  - Modified DES hashing algorithm - multiple rounds to slow down brute-force dictionary attacks
  - 1-way, but dictionary attacks now utterly trivial
    - 1,000,000 crypts/sec possible on high-end machine...
  - Many newer systems use MD5/sha1 hash (multiple rounds) - e.g., FreeBSD's MD5 is 10x slower than crypt
  - Most systems now have /etc/{shadow,master.passwd} -- only root can see crypted password
  - But it's still valuable - lets you get to other systems
- Seeds: Prevent pre-computed dictionary attacks
- Lesson: Parameterize hash function to evolve over time (new hashes, more rounds)

What are you protecting?

- Many vulnerabilities come from components that "weren't supposed" to be trusted
- e.g., address change bug (has happened multiple times in history of banking)
  - Change victim's address to known location
  - Send new ATM + PIN
  - Change address back
- Software examples: exec(prog, ..., user_supplied_in)
  - Does your "secure" program shell out to something else and pass it user-supplied input? Hope that something else doesn't have a buffer overflow/ etc.
- Requires deep understanding of dependencies of program
  - And controlling/ensuring that over time!

Trust

- Many attacks exploit trust; important to understand explicit and implicit trusts in your system
  - .rhosts, /etc/exports -- explicit
  - .rhosts may depend on DNS...
    - DNS may be administered by another person
    - Does that person understand effect of changing the mapping for trusted.host.com to another IP address?
  - All of your code trusts
    - The compiler, the OS, the {ruby,python,etc} interpreter, the 3rd party libraries you downloaded, ...

Too clever by half

- "The bulk of computer security R&D is of marginal relevance to real needs"
- "We found almost all attacks involved blunders, insiders, or both. High tech attacks were rare, and the 2 that did occur can be seen as effect of absence of security. However, high tech threats were what products were designed to prevent. And product complexity contributed to the blunders that caused real losses."
- "Continuity matters; and we do not really understand how to maintain effective control in an organization whose structure is constantly changing. … It can take many years for a security capability to mature and become effective."

Modern example

- example from {industry, government} in 2007 (person who was clueful and in security-sensitive area):
  - "I get a lot of security best practice mandates coming from above. Things like virus scanning, firewall configs, application configs, etc. Problem is, our single largest cause of security problems, by a huge margin, is lost laptops..."

Real sources of attacks

- Wonderful long list of real sources of attacks:
  - Program bugs: ATM code wrong 1 in x0,000 transactions
  - Postal/inperson interception: capture cards & PINs in mail
  - Insider thieves
    - Change address, issue a card, change it back
    - Put a laptop into an ATM and monitor acct numbers & PINs
    - Use same keys in test/training system as real system
    - Debugging backdoors left in production systems
    - Politics of cost savings weakening security processes
    - Code system to issue only a small number of different PINs
  - Other brute force attacks
    - Stand in line, watch PINs, get acct from receipts left behind
    - Offline checking of PINs used too simple an algorithm
    - Walk in and ask bank to change PIN, they probably will
    - Build your own ATM, get people to give it acct number and PIN

On the network

- On Internet today, un-firewalled host sees
  - Constant attempts to exploit prior & current vulnerabilities
    - Lots of them!
  - Brute-force password guesses of common account names
  - Attempts to use host as a proxy for other protocols {spamming, hide tracks, access data controlled by source address, etc.}
- Only one of these things has to go wrong[11]. :(

Robust & Explicit Security

- "This is a failure of certification. Staff don't have the knowledge. Product certification should be tied to skill of user."
- "Robust security means systems tolerate minor errors in design, implementation and operation without (much) loss of security"
- "Be explicit about assumptions of properties at interface of levels/modules; ie., naming, freshness, chaining"
  - Specs should list all possible failure modes, prevention strategies for each, implementation of each strategy, and a review process by an independent certification body
- Make deep pockets more responsible than shallow pockets so someone actually pays for security R&D & insurance

Commentary

- Byzantine faults are not exclusively those that are most academic/insidious (ie., brute crypto breaking)
  - Include all the common things that are possible
- Security should consider average case as well as worst case attacks
  - Threat models should include multiple concurrent failures
- Hiding in obscurity may just mean more damage is done before weakness is found
  - Expose security design, implementation, operation to extensive inspection and criticism (open src argument)

Security Breach Disclosure Regs

BILL NUMBER: SB 1386 CHAPTERED
BILL TEXT
CHAPTER 915
FILED WITH SECRETARY OF STATE SEPTEMBER 26, 2002
APPROVED BY GOVERNOR SEPTEMBER 25, 2002
PASSED THE SENATE AUGUST 30, 2002
PASSED THE ASSEMBLY AUGUST 26, 2002

INTRODUCED BY Senator Peace
(Principal coauthor: Assembly Member Simitian)

FEBRUARY 12, 2002

An act to amend, renumber, and add Section 1798.82 of, and to add Section 1798.29 to, the Civil Code, relating to personal information.

LEGISLATIVE COUNSEL'S DIGEST

SB 1386, Peace. Personal information: privacy.
Existing law regulates the maintenance and dissemination of personal information by state agencies, as defined, and requires each agency to keep an accurate account of disclosures made pursuant to specified provisions. Existing law also requires a business, as defined, to take all reasonable steps to destroy a customer's records that contain personal information when the business will no longer retain those records. Existing law provides civil remedies for violations of these provisions.
This bill, operative July 1, 2003, would require a state agency, or a person or business that conducts business in California, that owns or licenses computerized data that includes personal information, as defined, to disclose in specified ways, any breach of the security of the data, as defined, to any resident of California whose unencrypted personal information was, or is reasonably believed to have been, acquired by an unauthorized person. The bill would permit the notifications required by its provisions to be delayed if a law enforcement agency determines that it would impede a criminal investigation. The bill would require an agency, person, or business that maintains computerized data that includes personal information owned by another to notify the owner or licensee of the information of any breach of security of the data, as specified. The bill would state the intent of the Legislature to preempt all local regulation of the subject matter of the bill. This bill would also make a statement of legislative findings and declarations regarding privacy and financial security.

THE PEOPLE OF THE STATE OF CALIFORNIA DO ENACT AS FOLLOWS:

The Staw that Broke the Data Broker's Back

As a result of California Civil Code 1798 (commonly known as California Senate Bill 1386), organizations must disclose security incidents that involve private information about California residents, such as the ChoicePoint breach. Further California legislation - in what is knows as California Assembly Bill 1950 - has recently gone into effect that requires "reasonable security" controls around California resident data within organizations and their business partners.[2]

Importance of Little Things

- Social engineering example:
  - Step 1: Get org chart & contact info
  - Step 2: Call front-line employee, pretend to be some other employee (gives you legitimacy), ask if so-and-so is there (find someone who's out)
  - Step 3: Call someone else, say "this is person-who's-out; I'm sick and working from home today but needed to get something off my computer. Could you go {do something innocuous but actually dangerous--hooking up modem, etc.}
- Systems example:
  - Get "secure" (encrypted) /etc/passwd, mount dictionary attack
  - Knowing .csh history: gives lists of other hosts to target with likely same password
  - Knowing .ssh/known_hosts: Same as above, but big, automatically maintained list...

Reflections on Trusting Trust.

Thompson84: Ken Thompson, Comm. Of the ACM, vol 27, no 8, August 1984.

Step 1: programs writing programs

- A self-reproducing program
  - Compiled & run, it outputs its source
- Basically, source contains initialized static variables containing character strings that are the source

```
char s[ ] = {
        '\t',
        '0',
        '\n',
        '}',
        ';',
        '\n',
        '\n',
        '/',
        '*',
        '\n',
        (213 lines deleted)
        0
};

/*
 * The string s is a
 * representation of the body
 * of this program from '0'
 * to the end.
 */

main( )
{
        int i;

        printf("char\ts[ ] = {\n");
        for(i=0; s[i]; i++)
                printf("\t%d, \n", s[i]);
        printf("%s", s);
}
```

Step 2: bootstraping

- Bin2.1 (Src2.1) => Bin2.1
  - Compiler compiles itself
  - Lets change compiler source
- Bin2.1 (Src2.2) fails
  - Need to bootstrap modifications
- Bin2.1 (Src2.3) => Bin2.3
- Bin2.3 (Src2.2) => Bin2.2
- Bin2.2 (Src2.2) => Bin2.2
  - Intermediate, 2.3, is gone
  - "code remembers what it learned"

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
...
```

FIGURE 2.2. 2.1

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return('\v');
...
```

FIGURE 2.1. 2.2

```
...
c = next( );
if(c != '\\')
        return(c);
c = next( );
if(c == '\\')
        return('\\');
if(c == 'n')
        return('\n');
if(c == 'v')
        return(11);
...
```

FIGURE 2.3.

Step 3: concealing hacks

- Mod compiler 3.1 into 3.2
- What are pattern & bug?
  - Pattern is login program src
  - Bug is login src w/ trapdoor
- Code inspection finds this
- Mod compiler to 3.3
  - Pattern2 is compiler 3.1
  - Bug2 is compiler 3.3
  - If "hacked" binary 3.3 is installed, compiling 3.1 (on disc) gives 3.3
  - But, inspection sees compiler 3.1

```
compile(s)
char *s;
{
        ...
}
```

FIGURE 3.1.

```
compile(s)
char *s;
{
        if(match(s, "pattern")) {
                compile("bug");
                return;
        }
        ...
}
```

FIGURE 3.2.

```
compile(s)
char *s;
{
        if(match(s, "pattern1")) {
                compile ("bug1");
                return;
        }
        if(match(s, "pattern 2")) {
                compile ("bug 2");
                return;
        }
        ...
}
```

FIGURE 3.3.

Security Wakeup Call

- This is a Turing Award talk
  - No numeric eval; but construction is convincing
- Code for backdoors
  - won't be found by source inspection if system+src install disc comes with a false compiler binary
- "can't trust code you did not totally create yourself"
- Made a lot of people recognize maliciousness and cleverness in "hacking"

Voting Machines

That said

- It's not *just* humans/etc.
  - Crypto failures: 802.11 WEP is atrocious!
    - Wars have been won/lost b/c of crypto breaking
  - Protocol failures: (yes, WEP, but others)
  - Design failures
  - Implementation failures** -- buffer overflows, SQL injections, format string attacks, yadda, yadda, yadda.
  - 20 years of sendmail exploitation. :)
- Whatever the weak point is...

The point

- Security is a whole-system challenge
  - Crypto, protocols, design, impl
  - But also: UI, human factors
  - Weakest link...
- Robustness and defense-in-depth
  - Isolation, minimum privilege, simplicity, explicit assumptions and dependencies
  - Require multiple failures?
  - Make it hard for humans to do wrong things
- Solutions are hard; require technical, social, legal changes -- better programming tools, UI understanding, user education, laws for disclosure and prosecution, ...