Designing and Writing Secure Application Code

John Mitchell

General topics in this course

- Vulnerabilities
  - How criminals break into things
    - Circumvent security mechanisms (e.g., dictionary attack)
    - Use code for purpose it was not intended (buffer overflow)
- Defensive programming
  - Build *all* software with security in mind
  - Make sure video game is not a boot loader
- Security Mechanisms
  - Authentication
  - Access control
  - Network protocols

This lecture

Before you start building ...

- What are the security requirements?
  - Confidentiality (secrets remain secret)
  - Integrity (meaning preserved)
  - Availability
  - Accountability
- What threats are possible?
- Who do you trust / not trust?

Security = preserve properties against attack

10 Platitudes [Viega and McGraw]

- Secure the weakest link
- Practice defense in depth
- Fail securely
- Follow the principle of least privilege
- Compartmentalize
- Keep it simple
- Promote privacy
- Remember that hiding is hard
- Be reluctant to trust
- Use your community resources

Secure the weakest link

- Think about possible attacks
  - How would someone try to attack this?
  - What would they want to accomplish?
- Find weakest link(s)
  - Crypto library is probably pretty good
  - Is there a way to work around crypto?
    - Data stored in encrypted form; where is *key* stored?
- Main point
  - Do security analysis of the whole system
  - Spend your time where it matters

Defense in Depth / Fail securely

- Failure is unavoidable – plan for it
- Have a series of defenses
  - If an error not caught by one, caught by another
- Examples
  - Firewall + network intrusion detection
  - SSH + Tripwire

Be Reluctant to Trust

- Clients, servers should not trust each other
  - Both can get hacked
- Trusted code should not call untrusted code

Principle of least privilege

- Give only the minimum privilege
  - Needed for the task, For minimum amount of time
- Compartmentalize
  - Minimize damage possible from one module
- Examples
  - Sendmail runs as root
    - Root privilege needed to bind port 25
    - No longer needed after port bind established
      - But most systems keep running as root
      - Also need root privileges to write to user mailboxes
  - Will look at qmail for better security design

Keep It Simple

- Use standard, tested components
  - Don't implement your own cryptography
- Don't add unnecessary features
  - Extra functionality ⇒ more ways to attack
- Use simple algorithms that are easy to verify
  - A trick that may save a few instructions may
    - Make it harder to get the code right
    - Make it harder to modify and maintain code

Promote Privacy

- Discard information when no longer needed
  - No one can attack system to get information
- Examples
  - Don't keep log of old session keys
  - Delete firewall logs
  - Don't run unnecessary services (fingerd)

Remember that hiding secrets is hard

- Information in compiled binaries can be found
- Insider attacks are common
  - Companies spend time/money on firewalls
  - Firewalls do not protect against inside attack
- Security by obscurity doesn't work!!!

Use Your Community Resources

- Consult experts
- Allow public review
- Use software, designs that other have used

Example: Mail Transport Agents

- Sendmail
  - Complicated system
  - Source of many vulnerabilities
- Qmail
  - Simpler system designed with security in mind
  - Gaining popularity

Qmail written by Dan Bernstein, starting 1995
$500 reward for successful attack; no one has collected

Recent Sendmail Vulnerability

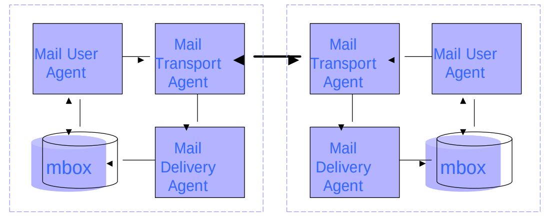
Sent: Tuesday, March 04, 2003 1:12 PM
To: unix-info@lists.Stanford.EDU
Subject: Stanford ITSS Security Alert: sendmail Header Processing Vulnerability

sendmail is the most popular Mail Transfer Agent (MTA) program in use on the Internet, …

sendmail contains an error in one of the security checks it employs on addresses in its headers, which may allow an attacker to execute malicious code within the sendmail security context, usually root…

All users of sendmail should patch immediately …

Simplified Mail Transactions



- Message composed using an MUA
- MUA gives message to MTA for delivery
  - If local, the MTA gives it to the local MDA
  - If remote, transfer to another MTA

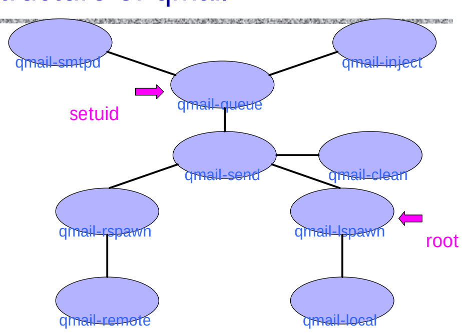
Example: qmail

- Compartmentalize
  - Nine separate modules
  - Each runs under different non-privileged UID: qmaild, qmailr, qmailq, … (except one as root)
  - If one module compromised, others not
    - SMTP server qmail-smtpd runs as user qmailr
    - Rest of the system runs as other users

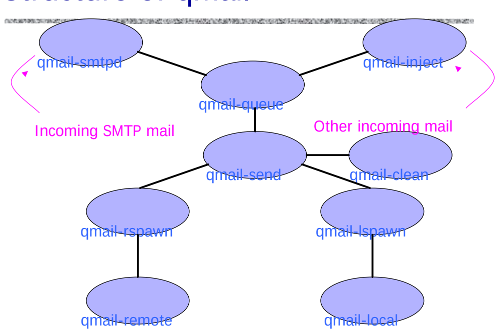
Example: qmail

- Least privilege
  - Each module uses least privileges necessary
  - Only one setuid program
    - setuid to one of the other qmail user IDs, not root
    - No setuid root binaries
  - Only one run as root
    - Spawns the local delivery program under the UID and GID of the user being delivered to
    - No delivery to root
    - Always changes effective uid to recipient before running user-specified program

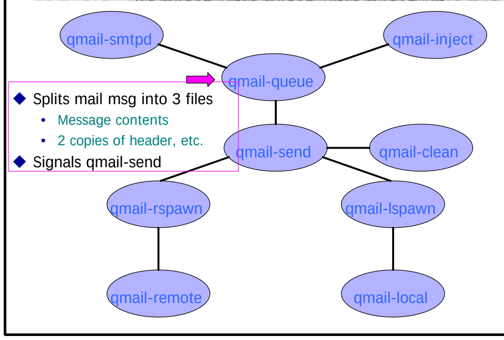
Structure of qmail



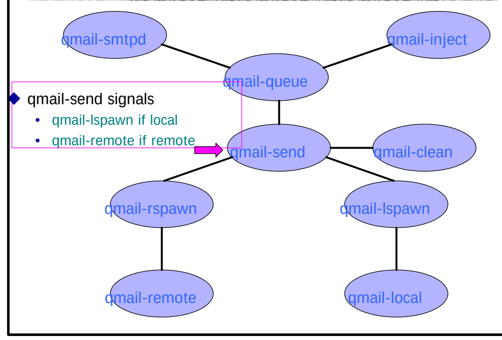
Structure of qmail



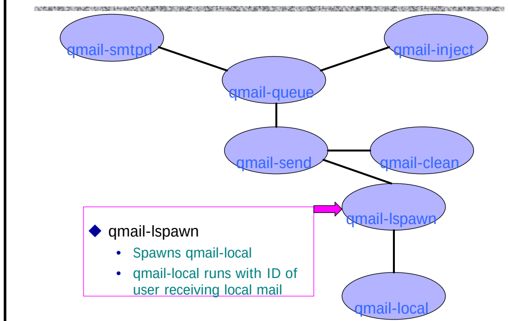
Structure of qmail



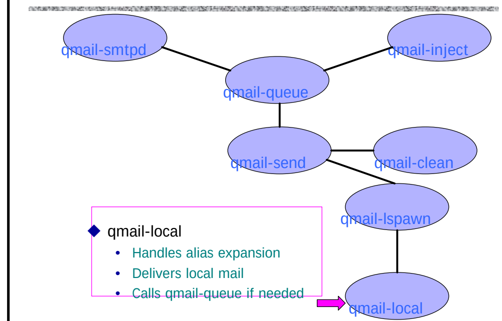
Structure of qmail



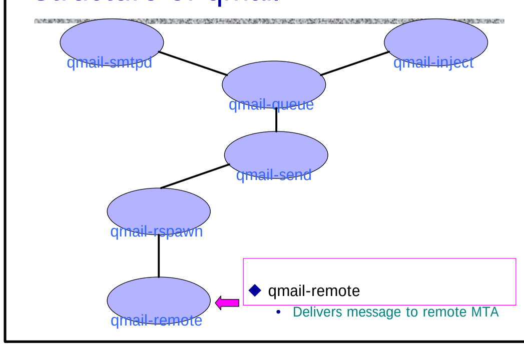
Structure of qmail



Structure of qmail



Structure of qmail



Principles, sendmail vs qmail

- Do as little as possible in setuid programs
  - Of 20 recent sendmail security holes, 11 worked only because the entire sendmail system is setuid
  - Only qmail-queue is setuid
    - Its only function is add a new message to the queue
- Do as little as possible as root
  - The entire sendmail system runs as root
    - operating system protection has no effect
  - Only qmail-start and qmail-lspawn run as root.

Principles, sendmail vs qmail

- Programs and files are not addresses
  - sendmail treats programs and files as addresses
    - "sendmail goes through horrendous contortions trying to keep track of whether a local user was responsible for an address. This has proven to be an unmitigated disaster" (DJB)
  - qmail programs and files are not addresses
    - "The local delivery agent, qmail-local, can run programs or write to files as directed by ~user/.qmail, but it's always running as that user. Security impact: .qmail, like .cshrc and .exrc and various other files, means that anyone who can write arbitrary files as a user can execute arbitrary programs as that user. That's it." (DJB)

Keep it simple

- Parsing
  - Limited parsing of strings
    - Minimizes risk of security holes from configuration errors
- Libraries
  - Avoid standard C library, stdio
    - "Write bug-free code" (DJB)
- Don't repeat functionality
  - One simple mechanism handles forwarding, aliasing, and mailing lists (instead of 3)
  - Single delivery mode instead of a selection

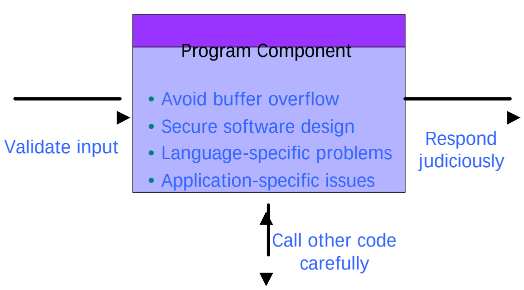
Keep it simple

	Lines	Words	Chars	Files
qmail-1.01	16028	44331	370123	288
sendmail-8.8.8	52830	179608	1218116	53
zmailer-2.2e10	57595	205524	1423624	227
smail-3.2	62331	246140	1701112	151
exim-1.90	67778	272084	2092351	127

Secure Implemention

- Generalities apply to design, implementation
  - Example: defense in depth within code
    - Check privileges several times
    - Helps if code is modified
    - Also can be useful if there is a race condition ...
- Once you have a design, implement carefully
  - Keep security issues in mind
  - Some tools can help

General categories [Wheeler]



See Wheeler's book on web

Tools for producing secure code

- C vs type safe languages
  - Buffer overflows are array bounds violations
  - Java, ML, ... check array bounds, prevent overflow
- Purify
  - Find memory errors on the heap
- Perl tainting
  - Track use of untrusted input
- Automated code analysis tools
  - Can catch many kinds of errors

Purify

◆ Goal
- Instrument a program to detect run-time memory errors (out-of-bounds, use-before-init) and memory leaks

◆ Technique
- Works on relocatable object code
  – Link to modified malloc that provides tracking tables
- Memory access errors: insert instruction sequence before each load and store instruction
- Memory leaks: GC algorithm

Perl tainting

◆ Run-time checking of Perl code
- Perl used for CGI scripts, security sensitive
- Taint checking stops some potentially unsafe calls

◆ Tainted strings
- User input, Values derived from user input
- Except result of matching against untainted string

◆ Prohibited calls
- print $form_data{"email"} . "\n";
  – OK since print is safe (???)
- system("mail " . $form_data{"email"});
  – Flagged system call with user input as argument

Safe Perl mail command (?)

◆ Check email string against pattern and parse

```
$email = $form_data{"email"};
if ( $email =~ /(\w{1}[\w-.]*)\@([\w-.]+)/) {
    $email = "$1\@$2";
} else { warn ("TAINTED DATA SENT BY ...");
    $email = ""; # successful match did not occur }
```

◆ What does this accomplish?
- Only send email to address that “looks good”
- Programmer responsible for “good” pattern
- Perl cannot guarantee that email addr is OK

Automated code analysis for C

◆ Example tool
- Ken Ashcraft and Dawson Engler, Using Programmer-Written Compiler Extensions to Catch Security Holes, IEEE Security and Privacy 2002
- Used modified compiler to find over 100 security holes in Linux and BSD
- http://www.stanford.edu/~engler/

◆ Benefit
- Capture recommended practices, known to experts, in tool available to all

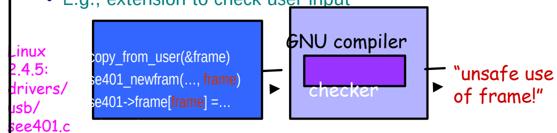
Checking secure software

◆ Many rules for writing secure code
- “sanitize user input before using it”
- “check permissions before doing operation X”

◆ How to find errors?
- Formal verification
  + rigorous
  – costly, expensive. *Very* rare to do for software
- Testing:
  + simple, few false positives
  – requires running code: doesn’t scale & can be impractical
- Manual inspection
  + flexible
  – erratic & doesn’t scale well.
- What to do??

Metacompilation (MC)

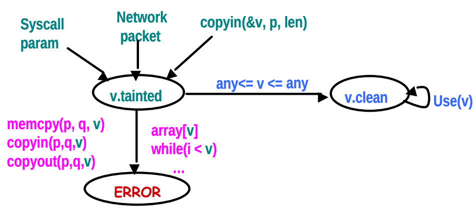
◆ Analyze compiler data structure to check code
- Extensions dynamically linked into GNU gcc compiler
- Applied down all paths in input program source
- E.g., extension to check user input



Actual error in Linux: raid5 driver disables interrupts, and then if it fails to allocate buffer, returns with them disabled. This kernel deadlock is actually hidden by an immediate segmentation fault since the callers dereference the pointer without checking for NULL

Sanitize integers before use

Warn when unchecked integers from untrusted sources reach trusting sinks



Linux: 125 errors, 24 false; BSD: 12 errors, 4 false

Example security holes

- Remote exploit, no checks

```
/* 2.4.9/drivers/isdn/act2000/capi.c:actcapi_dispatch */
isdn_ctrl cmd;
...
while ((skb = skb_dequeue(&card->rcvq))) {
   msg = skb->data;
   ...
   memcpy(cmd.parm.setup.phone,
          msg->msg.connect_ind.addr.num,
          msg->msg.connect_ind.addr.len - 1);
```

Example security holes

- Missed lower-bound check:

```
/* 2.4.5/drivers/char/drm/i810_dma.c */
if(copy_from_user(&d, arg, sizeof(arg)))
   return -EFAULT;
if(d.idx > dma->buf_count)
   return -EINVAL;
buf = dma->buflist[d.idx];
Copy_from_user(buf_priv->virtual, d.address, d.used);
```

User-pointer inference

- Problem: which are the user pointers?
  - Hard to determine by dataflow analysis
  - Easy to tell if kernel *believes* pointer is from user!
- Belief inference
  - "*p" implies safe kernel pointer
  - "copyin(p)/copyout(p)" implies dangerous user ptr
  - Error: pointer p has both beliefs.
- Implementation: 2 pass checker
  - inter-procedural: compute all tainted pointers
  - local pass to check that they are not dereferenced

Results for BSD and Linux

- All bugs released to implementers; most serious fixed

	Linux		BSD	
Violation	Bug	Fixed	Bug	Fixed
Gain control of system	18	15	3	3
Corrupt memory	43	17	2	2
Read arbitrary memory	19	14	7	7
Denial of service	17	5	0	0
Minor	28	1	0	0

	Linux	BSD
Local bugs	109	12
Global bugs	16	0
Bugs from inferred ints	12	0
False positives	24	4
Number of checks	~3500	594

Conclusions

- Security takes extra effort
  - Know your security goals
  - Design with security in mind
    - Compartmentalize, least privilege
    - Minimize setuid, root
  - Implement carefully
    - Keep it simple
    - Think about attacks; secure the weakest link
  - Use tools that detect common coding problems
    - There are also tools that can analyze designs, but that's another story (harder to use, current research)