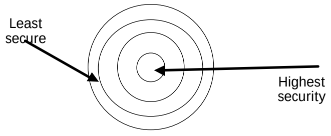
Software Security Holes and Defenses

Design of a secure system

- Follows a ring design.
  - Every *object* has an associated security attribute.
  - Every *subject* has a security clearance.

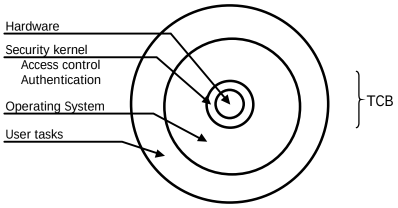


Least secure

Highest security

- Restricted interaction between rings.

Example: trusted OS



Hardware

Security kernel
Access control
Authentication

Operating System

User tasks

TCB

- TCB: part of the OS trusted to enforce security policy.

Bell-La Pdula Model

- Set of objects: O. Set of subjects: S.
- Each $o \in O$ and $s \in S$ has a security class C(o) and C(s).
- Property I: subj. s may have *read* access to obj. o only if: $C(o) \le C(s)$.
- Property *: subj. s who has read access to obj. o may have write access to obj. p only if: $C(o) \le C(p)$.
- Model errors on safety.

Evaluation: the orange book

- Department of Defense, 1979:
  Trusted Computer System Evaluation Criteria.
- Ratings:
  - D: Minimal protection. Anyone can get this rating.
  - C1: Discretionary security. **Users can disable sec. mech.**
  - C2: Controlled access. Per user protection. Discretionary.
  - B1: Labeled protection. **Every object labeled**. Bell-La Padula
  - B2: Structured protection. More OS module verification.
  - B3: Security domains. Modular OS design. Clear sec. policy.
  - A1: Verified design. Formally verified system design.
- Example: NT is considered C2 compliant.

Buffer Overflow Attacks

Buffer overflows

- Extremely common bug.
- Over 50% of all CERT advisories:
  - 1997: 16 out of 28 CERT advisories.
  - 1998: 9 out of 13 -”-
  - 1999: 6 out of 12 -”-
- Often leads to total compromise of host.
  - Fortunately: exploit requires expertise and patience.
  - Two steps:
    - – Locate buffer overflow within an application.
    - – Design an exploit.

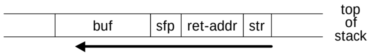
What are buffer overflows?

- Suppose a web server contains a function:

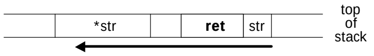
```
void func(char *str) {
  char buf[128];

  strcpy(buf, str);
}
```

- When the function is invoked the stack looks like:

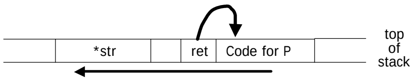


- What if **`*str`** is 136 bytes long? After **`strcpy:`**



Basic stack exploit

- Main problem: no range checking in strcpy().
- Suppose *str is such that after strcpy stack looks like:



Program P: **`exec( “/bin/sh” )`**

- When func() exits, the user will be given a shell !!
- Note: attack code runs *in stack*.
- To determine ret guess position of stack when func() is called.

Exploiting buffer overflows

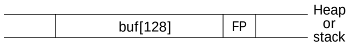
- Suppose web server calls func() with given URL.
- Attacker can create a 200 byte URL to obtain shell on web server.
- Some complications:
  - Program P should not contain the ‘\0’ character.
  - Overflow should not crash program before func() exists.
- Recent buffer overflows of this type:
  - Overflow in MIME type field in MS Outlook.
  - Overflow in ISAPI in IIS.

More general exploits

- Basic stack exploit can be prevented by marking stack segment as non-executable.
  - Code patches exist for Linux and Solaris.
  - Does not block more general overflow exploits.
- General buffer overflow exploits are based on two orthogonal steps:
  - Arrange for attack code to be present in program space.
  - Cause program to execute attack code.

Causing program to exec attack code

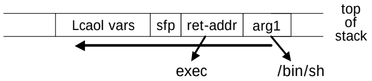
- Stack smashing attack:
  - Override return address in stack activation record by overflowing a local buffer variable.
- Function pointers: (used to attack Linux superprobe)



  - Overflowing buf will override function pointer.
- Longjmp buffers: longjmp(pos) (used to attack Perl 5.003)
  - Overflowing buf next to pos overrides value of pos.

Placing attack code in program

- Injecting attack code:
  - Place code in stack variable (local vars).
  - Place code in a heap variable (malloc'ed vars).
  - Place code in static data segment (static vars).
- Using existing code: the libc exec function.
  - Cause FP or ret-addr to point to libc exec func.
  - At same time override first argument to be \bin\sh



Finding buffer overflows

- Hackers find buffer overflows as follows:
  - Run web server on local machine.
  - Issue requests with long tags.
    All long tags end with "$$$$$".
  - If web server crashes,
    search core dump for "$$$$$" to find
    overflow location.
- Some automated tools exist. (eEye Retina, ISIC).

Preventing buf overflow attacks

- Main problem:
  - strcpy(), strcat(), sprintf() have no range checking.
  - "Safe" versions strncpy(), strncat() are often misleading
    - strncpy() may leave buffer unterminated.
    - strncpy(), strncat() encourage off by 1 bugs.

strncpy(dest, src, strlen(src)+1)

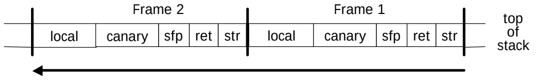
- Defenses:
  - Static source code analysis.
  - Run time checking.
  - Black box testing (e.g. eEye Retina, ISIC).

Static source code analysis

- Statically check source to detect buffer overflows.
  - Several consulting companies.
- Can we automate the review process?
- Several tools exist:
  - @stake.com (l0pht.com): SLINT (designed for UNIX)
  - rstcorp: its4. Scans function calls.
  - Berkeley: Wagner, et al. Tests constraint violations.

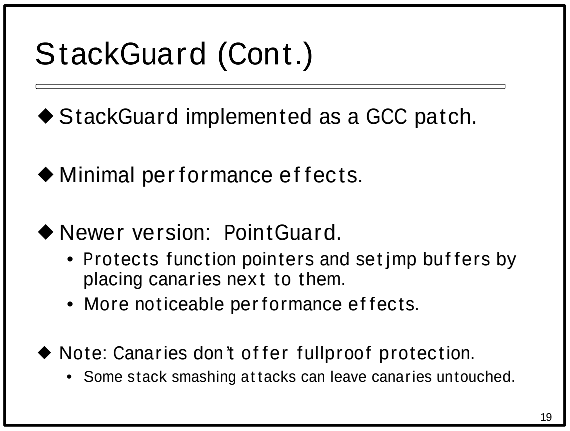
Run time checking: StackGuard

- Solution 1: Runtime range checking
  - Significant performance degradation.
  - Hard for languages like C and C++.
- Solutions 2: StackGuard (OGI)
  - Run time tests for stack integrity.
  - Embed "canaries" in stack frames and verify their integrity prior to function return.



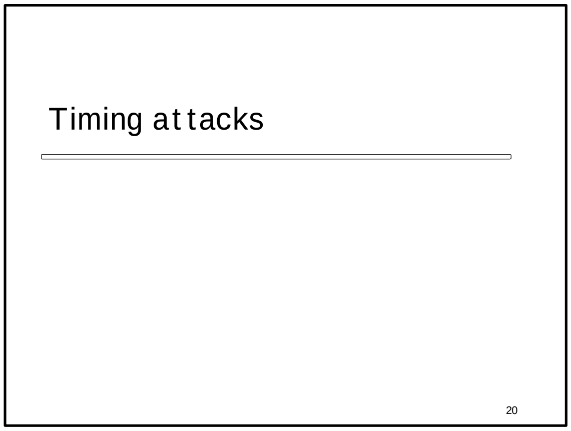
Canary Types

- Random canary:
  - Choose random string at program startup.
  - Insert canary string into every stack frame.
  - Verify canary before returning from function.
  - To corrupt random canary attacker must learn current random string.
- Terminator canary:
  Canary = 0, newline, linefeed, EOF
  - String functions will not copy beyond terminator.
  - Hence, attacker cannot use string functions to corrupt stack.

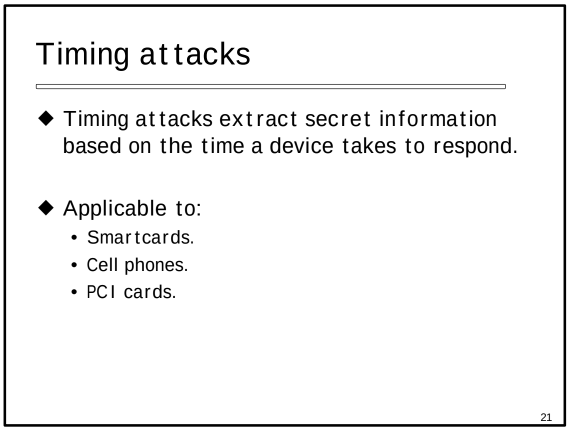


StackGuard (Cont.)

- StackGuard implemented as a GCC patch.
- Minimal performance effects.
- Newer version: PointGuard.
  - Protects function pointers and setjmp buffers by placing canaries next to them.
  - More noticeable performance effects.
- Note: Canaries don't offer fullproof protection.
  - Some stack smashing attacks can leave canaries untouched.

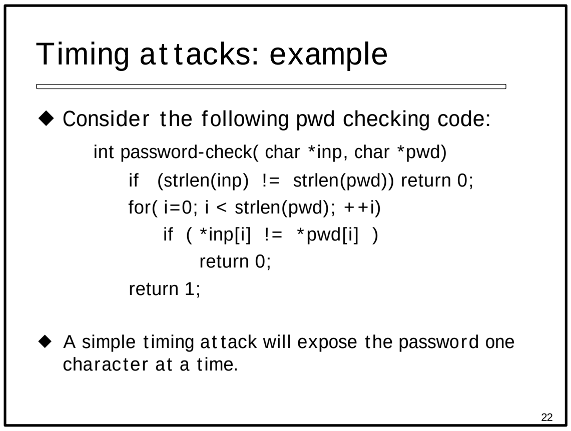


Timing attacks



Timing attacks

- Timing attacks extract secret information based on the time a device takes to respond.
- Applicable to:
  - Smartcards.
  - Cell phones.
  - PCI cards.

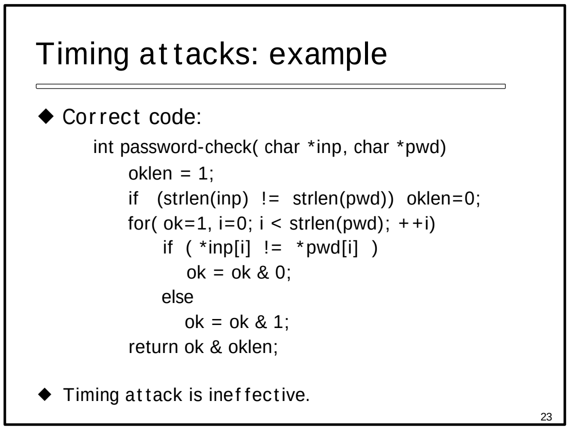


Timing attacks: example

- Consider the following pwd checking code:

```
int password-check( char *inp, char *pwd)
     if   (strlen(inp)  !=  strlen(pwd)) return 0;
     for( i=0; i < strlen(pwd); ++i)
          if  ( *inp[i]  !=  *pwd[i]  )
               return 0;
     return 1;
```

- A simple timing attack will expose the password one character at a time.

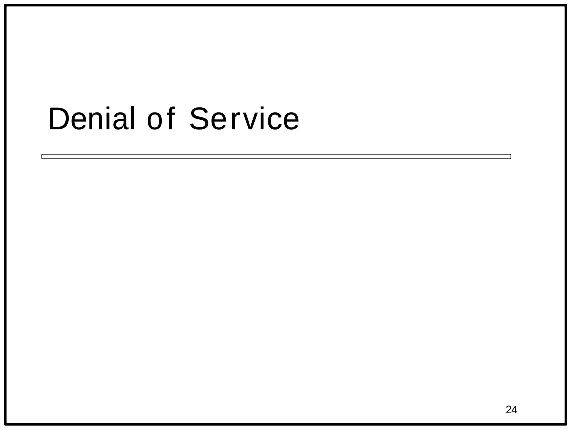


Timing attacks: example

- Correct code:

```
int password-check( char *inp, char *pwd)
     oklen = 1;
     if   (strlen(inp)  !=  strlen(pwd))  oklen=0;
     for( ok=1, i=0; i < strlen(pwd); ++i)
          if  ( *inp[i]  !=  *pwd[i]  )
              ok = ok & 0;
          else
              ok = ok & 1;
     return ok & oklen;
```

- Timing attack is ineffective.



Denial of Service

Denial of Service (DoS)

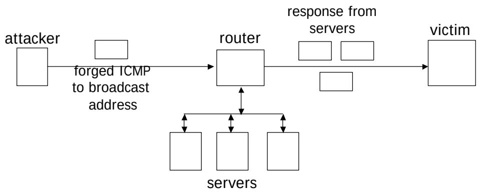
- Disabling a service by consuming resources.
- Example: Apache web server.
  - Apache runs N preforked processes to handle incoming connections.
  - Attacker: open N very slow long lived connections to web server.
  - All Apache processes will serve slow connections. No new connections will be served.
  - Solution: secure connection mgmt, e.g. Ingrian.

Distributed Denial of Service

- Using multiple hosts to launch Denial of Service attacks.
- Widely available DDoS tools:
  - Smurf
  - Trinoo
  - Trible Flood Network (TFN, TFN2K)
  - Stacherldraht
  - Shaft
  - Mstream
  - …

Smurf

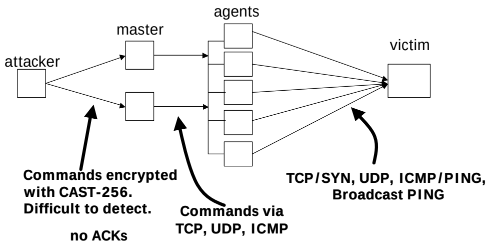
- Send ICMP packet with forged origin IP.
  - All machines that receive packet respond to victim.



- Router or firewall should be configured to block such packets.

Tribal Flood Network (TFN2K)

- Coordinated distributed attack.
  - Much harder to detect and prevent.



Defenses

- Constantly test if local machines became DDoS agents (e.g. TFN agents).
  - FBI publishes tools to detect known agents.
  - Cat and mouse game…
- Much work on detecting attack origin:
  - Savage et al.: routers embed info in packets. Victim can slowly piece together attack origin.
  - Burch, Cheswick: controlled flooding of subnets.
  - Bellovin: routers sign random fraction of packets.

Covert channels

- Bell-La padula: prevent subjects with different access rights from communicating.
  - Problem: covert channels.
- Covert channels:
  - communication channels undetected by the security policy enforcer.
- Example: **File locking**:
  - High clearance subject frequently locks and unlocks a file.
  - Low clearance subject checks lock status.
  - Using synchronized timer: 1000bit/sec transfer rate.

Covert channels using DNS

- Java security manager:
  - Prevents applets from communicating with most hosts.
  - Uses DNS to get IP address of requested hostname.
- Covert channel: (Dean96)
  - Applet frequently attempts to communicate with hosts: attack0nnn.com or attack1nnn.com
  - By monitoring DNS attacker reads information.

References

- Buffer overflows: attacks and defenses for the vulnerability of the decade. http://www.immunix.org/StackGuard/discex00.pdf
- A first step towards automated detection of buffer overrun vulnerabilities. http://www.cs.berkeley.edu/~daw/papers/overruns-ndss00.ps
- Smashing the stack for fun and profit. www.phrack.com Article p49-14. By Aleph1
- Bypassing StackGuard and StackShield. www.phrack.com Article p56-6. By Bulba and Kil3r
- Distributed denial of service attacks/tools. http://staff.washington.edu/dittrich/misc/ddos
- Practical network support for IP traceback. http://www.cs.washington.edu/home/savage/traceback.html