Announcements

• Homework 3 will be available on the web site this afternoon.

• Project 2 is graded
  – Mean: 89
Recap

• Capability Lists
• Firewalls

• Today
  – Firewalls
  – Program Security (Buffer Overflows)
When to Filter

Router

Inside

Outside
On Input or Output

• Filtering on output can be more efficient since it can be combined with table lookup of the route.

• However, some information is lost at the output stage
  – e.g. the physical input port on which the packet arrived.
  – Can be useful information to prevent address spoofing.

• Filtering on input can protect the router itself.
Recommend: Filter ASAP

<table>
<thead>
<tr>
<th>Action</th>
<th>src</th>
<th>port</th>
<th>dest</th>
<th>port</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>BAD</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>we don’t trust them</td>
</tr>
<tr>
<td>allow</td>
<td>*</td>
<td>*</td>
<td>GW</td>
<td>25</td>
<td>connect to our SMTP</td>
</tr>
<tr>
<td>allow</td>
<td>GW</td>
<td>25</td>
<td>*</td>
<td>*</td>
<td>our reply packets</td>
</tr>
</tbody>
</table>

Is preferred over:

<table>
<thead>
<tr>
<th>Action</th>
<th>src</th>
<th>port</th>
<th>dest</th>
<th>port</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>*</td>
<td>*</td>
<td>BAD</td>
<td>*</td>
<td>subtle difference</td>
</tr>
<tr>
<td>allow</td>
<td>*</td>
<td>*</td>
<td>GW</td>
<td>25</td>
<td>connect to our SMTP</td>
</tr>
<tr>
<td>allow</td>
<td>GW</td>
<td>25</td>
<td>*</td>
<td>*</td>
<td>our reply packets</td>
</tr>
</tbody>
</table>
Example of a Pitfall

- Filter output to allow incoming and outgoing mail, but prohibit all else.

<table>
<thead>
<tr>
<th>Action</th>
<th>dest</th>
<th>port</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>allow</td>
<td>*</td>
<td>25</td>
<td>incoming mail</td>
</tr>
<tr>
<td>allow</td>
<td>*</td>
<td>&gt;= 1024</td>
<td>outgoing responses</td>
</tr>
<tr>
<td>block</td>
<td>*</td>
<td>*</td>
<td>nothing else</td>
</tr>
</tbody>
</table>

- Apply this output filter set to both interfaces of the router. Does it work?

- Unintended consequence: allows all communication on high numbered ports!
Circuit Level Gateways

- Caller connects to a TCP port on the gateway and the gateway connects to a TCP port on the other side. It relays bytes, acting like a wire.
- More general-purpose than application-level but allows finer control than filtering only.
- Example: valuable logs of connections can be kept.
Application Level Gateways

- The gateway acts as an intermediary at the level of the application, receiving outgoing commands, relaying them, obtaining responses and relaying them back to the source.
- Mail gateways are a typical example.
- Very strong control over security, but
- Special purpose software is required.
Program Security

• Firewalls are not enough!
  – Some data & code is intentionally permitted through the firewall

• Programs have vulnerabilities
  – Exploitable bugs in “good” programs
  – Too many “features” (i.e. automatically execute macros)

• Malicious programs
  – Viruses
  – Worms
  – Trojan Horses
Buffer Overflow Attacks

• > 50% of security incidents reported at CERT are due to buffer overflow attacks

• Problem is access control but at a very fine level of granularity

• C and C++ programming languages don’t do array bounds checks
C’s Control Stack

```c
f() {
    g(parameter);
}

g(char *args) {
    int x;
    // more local
    // variables
    ...
}
```

Diagram:
- **f’s stack frame**
  - input parameter
  - return address
  - base pointer
  - local variables
  - more local variables

Larger Addresses
C’s Control Stack

```c
f() {
    g(parameter);
}

g(char *args) {
    int x;
    // more local
    // variables
    ...
}
```
C’s Control Stack

f() {
  g(parameter);
}

g(char *args) {
  int x;
  // more local variables
  ...
}

int x;
// local variables
return address
Input parameter
f’s stack frame
base pointer
Larger Addresses
Buffer Overflow Example

```c
int g(char *text) {
    char buffer[128];
    strcpy(buffer, text);
    return 0;
}
```

- **buffer[]**
- **base pointer**
- **return address**
- **Attack code 128 bytes**
- **ADDR**
- **f’s stack frame**
- **text**
- **f’s stack frame**

CSE331 Fall 2002
Buffer Overflow Example

```c
void g(char *text) {
    char buffer[128];
    strcpy(buffer, text);
}
```

The diagram illustrates the memory allocation and the attack code being written into the buffer. The function `g` takes a pointer to a string and copies it into a buffer of 128 bytes. If the input string is longer than 128 bytes, it can cause a buffer overflow, potentially leading to security vulnerabilities.
The Problem

• C’s `strcpy`, `gets`, `strcat` functions don’t check array bounds

• These functions are used extensively
  – Feb. 25 2002: Internet Explorer HTML tags
    `<EMBED SRC="ATTACK CODE">`
  – telnetd Unix Telnet Daemon
  – Microsoft Outlook
  – …
Solutions

• Don’t write code in C
  – Use a safe language instead (Java, C#, …)
  – Not always possible (low level programming)
  – Doesn’t solve legacy code problem

• Link C code against safe version of libc
  – May degrade performance unacceptably

• Software fault isolation
  – Instrument executable code to insert checks

• Program analysis techniques
  – Examine program to see whether “tainted” data is used as argument to strcpy