Secure – Programming - Scenario

Eine Einführung in das Ausnutzen von Schwachstellen in Programmen und den Schutz vor derartigen Attacken

spranger@informatik.hu-berlin.de

Gliederung

- Einleitung
- Schwachstellen
  - Buffer Overflows
    - stack
    - heap
  - Malloc – Probleme
  - Format String
- Zusammenfassung/Ausblick

Einleitung

- Was ist drin im Vortrag?
  - Einleitung in das Thema der Verwundbarkeit von Programmen am Beispiel von C/C++ Programmen
  - Vorstellen von Mechanismen der Ausnutzung solcher Verwundbarkeiten
  - Vorstellen von Tools die das Auffinden von Schwachstellen ermöglichen
  - Vorstellen von Schutzmechanismen und Schutzwerkzeugen
  - x86, linux

Einleitung

- Was ist dieser Vortrag nicht?
  - keine Einleitung zu Problemen die in anderen Programmiersprachen oder Technologien auftreten
    - Web (CGI, PHP, Perl)
    - Java
    - SQL (SQL-Injection)
  - keine umfassende Einführung in Exploit-Design
  - state of the art

Einleitung

- Zahlen (www.cert.org)
  - gemeldete Schwachstellen

2000	1090
2001	2437
2002	4129
2003	3784
1Q-2Q 2004	1740

Einleitung

- Zahlen (Forts.)
  - gemeldete Sicherheitsvorfälle

1999	9859
2000	1999
2001	52658
2002	82094
2003	137539

Buffer Overflows

- Einleitung
- Stack based attacks
- Heap/bss based attacks
- Schutzmechanismen/Werkzeuge

Buffer Overflows - Einführung

- 1988 erstmalig benutzt (fingerd)
- führt häufig zur kompletten Übernahme des Systems oder zu nicht abfangbaren Systemcrashs
- häufiges Problem bei C/C++ - Programmen
- Zahlen

1997	16 von 28 Schwachstellen
1998	9 von 13 Schwachstellen
1999	6 von 12 Schwachstellen

Buffer Overflows - Einführung

- Was ist ein Buffer Overflow?
  - Überschreiben eines Puffers (buffer) fester Länge mit mehr Werten als der Puffer (buffer) halten kann
  - Buffer in C/C++ sind z.B. Arrays
- Wie kommt es zu Buffer Overflows?
  - Fehlende Überprüfung der Länge eines zu beschreibenden Buffers
  - vorwiegend durch libc Funktionen im Zusammenhang mit strings

Buffer Overflows - Einführung

- Verwundbare Funktionen
  - alle die auf Arrays(insbes. auf strings) arbeiten ohne die Länge explizit als Argument zu verwenden
  - strcpy, sprintf, strcat, gets, scanf
- Arten
    - stack buffer overflows
    - heap/bss buffer overflows

Buffer Overflows – Stack based

- Verwundbarer Puffer liegt auf dem Stack
- *Beispiel (1)*

```
void function(char* attack_buffer)
{
    char vuln_buffer[4];
    strcpy(vuln_buffer, attack_buffer);
}
main(int argc, char** argv)
{
    function(argv[1]);
}
```

Buffer Overflows – Stack based

- Problem:
  - auf dem Stack sind gleichzeitig auch Verwaltungsinformationen (Rücksprungadresse)
  - Überschreiben mit Boguswerten führt zu Programmabsturz
  - Überschreiben mit cleveren Werten führt zur Ausführung beliebiger Anweisungen
- *Beispiel(2) folgt*

Buffer Overflow – Stack based

```
void function(char*){ ...}
int main()
{
        int x;
        char* attack_buffer = ...;
        x=0;
        function(attack_buffer);
        x=1;
        printf("%d\n",x);
        return 0;
}
```

Buffer Overflows – Stack based

- Ausnutzung der Schwachstelle
  - Überschreiben der Rücksprungadresse mit einem Wert der auf eigenen Anweisungen zeigt
    - im Puffer selber
    - in anderen Puffern
    - in Umgebungsvariablen (env)
  - Überschreiben der Rücksprungadresse mit einer Library-Funktionsadresse (system)

Buffer Overflow – Stack based

- Folge
  - der eigene Code läuft im Kontext des angegriffenen Programms
  - vollständige Übernahme des Prozesses
- *Beispiel (3)*

Buffer Overflow – heap/bss based

- der verwundbare Puffer liegt in der heap oder .bss sektion des Programms
- d.h. kein direkter Zugriff auf die Returnadresse der Funktion oder EIP
- aber Überschreiben von beliebigen Daten in der entsprechenden Sektion
- mögliche Ziele:
  - Pointer auf Strings mit Dateinamen
  - memory managment information (s. Malloc-Probleme)

Buffer Overflow – statischer Schutz

- statische Quellcodeanalyse
  - manuell
  - automatische Tools
    - RATS, ITS *(Beispiel)*
  - Austausch verwundbarer Funktionen
- brute forcing of arguments
  - bfbtester

Buffer Overflow – dynamischer Schutz

- Canary
  - random canary
    - Zufallssequenz am Programmstart erzeugen/auswählen
    - in jeden Stackframe einfügen (oder heap-chunk)
    - vor Verlassen der Funktion canary überprüfen

Buffer Overflow – dynamischer Schutz

- Canary (Forts.)
  - Beispiele
    - StackGuard/PointGuard (GCC-Patch)
    - ssp/ProPolice (OpenBSD)
    - /GS – Microsoft Compiler Option
  - Probleme
    - Programme müssen neu übersetzt werden
    - Programm wird beendet
    - in der Regel nur der Stack geschützt
    - Performance

Buffer Overflow – dynamischer Schutz

- Non-exec-stack
  - Markieren der Stack-Sektion als nicht ausführbar
  - Beispiele
    - OpenWall (Solar Designer kernel patch)
    - exec-shield (Redhat)
    - PaX (markieren von Seiten als nicht exec Stack, Heap, bss)
    - Hardware

Buffer Overflow – dynamischer Schutz

- Non-exec-stack(Forts.)
  - Probleme
    - der Attackcode (Payload) muss nicht auf dem Stack liegen (dynamische Buffer, libc calls)
    - Programm wird beendet
    - z.T. nur der Stack geschützt

Buffer Overflow – dynamischer Schutz

- libsafe
  - dynamisch geladene Bibliothek
  - fängt libc-calls
  - überprüft bei verwundbaren libc-Funktionen, ob Platz (framepointer – dest) auf dem Stack ist
  - Probleme
    - nur x86
    - Schutz nur für library-Funktionen
    - Programm wird beendet

Buffer Overflow – dynamischer Schutz

- andere Möglichkeiten
  - Return Adress Obfuscation (xor)
  - libc function adress Randomization

Gliederung

- Einleitung
- Schwachstellen
  - Buffer Overflows
    - stack
    - heap
  - Malloc – Probleme
  - Format String
- Zusammenfassung/Ausblick

Malloc Probleme - Einführung

- Was ist malloc?
  - malloc, realloc, free – sind Funktionen zur dynamischen Speicherverwaltung (heap)
  - setzen auf brk() auf (Systemruf der die Heap – Größe verändert)
  - malloc – Interface teilt den großen Block(in chunks), setzt Speicher frei, verhindert Fragmentation
  - wenige Implementationen: gnu, System V, Microsoft (rtlHeap)

Malloc Probleme - Einführung

- Problem
  - da auch die Speicherinformations -verwaltungsdaten dynamisch sind, werden sie in der Regel mit auf dem Heap abgelegt (in-band)
- hier nur gnu libc betrachtet
- Layout eines chunks

Malloc Probleme - Einführung

- size hat spezielle Bedeutung
  - wenn malloc aufgerufen wird, wird 4 zur Größe addiert und dann aligned (padding)
  - d.h. die kleinsten 3 bit sind unbenutzt
  - wobei das letzte davon als Flag (INUSE_PREV) benutzt wird
- Freigabe von Speicher
  - eventuelles mergen mit Nachbarn
  - einhängen in Freispeicherliste

Malloc Probleme - Einführung

- neues layout

- Konsequenz
  - Speicherverwaltungsinformationen sind beeinflußbar

Malloc Probleme - Angriff

- Angriff mit Hilfe eines Makros, daß in free benutzt wird
- unlink(P, BCK, FCK)

```
#define unlink(P, FD, BD)
{
    BD = P->bk;
    FD = P->fd;
    FD->bk = BD;
    BK->fd = FD;
}
```

Malloc Probleme - Angriff

- es werden also Pointer aus dem Speicher genommen und an deren Inhalt mit Werten aus dem Speicher geschrieben
- z.B.

```
fd = retloc - 12;
bk = retaddr;
```

Gliederung

- Einleitung
- Schwachstellen
  - Buffer Overflows
    - stack
    - heap
  - Malloc – Probleme
  - Format String
- Zusammenfassung/Ausblick

Format String Bug - Gliederung

- Einleitung
- Angriffe
- Schutz

Format String Bug - Einführung

- Was ist ein Format String?
  - Char – Array mit inline Formatanweisungen
  - *Beispiel*

```
{ ...
    int i = 0;
    printf("int i:%d\n", i);
...}
```

  - vor allem in der Printf – Funktionsfamilie benutzt (printf, scanf, fprintf...)
  - aber auch bei syslog, err, warn
  - Form: '%' [minimal width] 'd'|'s'|'x'|'n'...

Format String Bug - Einführung

- Wie werden Format-Strings verarbeitet?
  - durch Funktionen mit variablen Argumentmengen
  - es gibt einen internen Pointer, der auf das nächste Argument zeigt
  - dieser Pointer wird weitergezählt, wenn ein Argument eingefügt wurde
  - d.h. der String wird von vorne nach hinten ausgegeben, trifft man unterwegs auf '%' wird das Argument entsprechend der Formatierungsanweisung ausgegeben

Format String Bug – Angriffe

- Beenden von Programmen (SIGSEGV, core dump)
  - durch illegale Pointer (%s)
  - *Beispiel (4)*
- Lesen des Stack
  - durch Zugriff auf folgende Speicherelemente (z.B. mit %d)
  - *Beispiel (5)*

Format String Bug - Angriffe

- Lesen von Speicher (arbitrary)
  - durch forwarding des Stack-Pointers zum Format-String selber
  - dafür muss der Format String auf dem Stack sein
  - *Beispiel (6)*

Format String Bug - Angriffe

- Überschreiben von Speicher
  - "%n" schreibt in ein folgenden int* die Anzahl der bereits geschriebenen Character (*Beispiel (7))*
  - die Adresse an die geschrieben wird, muss irgendwie auf dem Stack liegen
  - am einfachsten liegt der Format String selber auf dem Stack (wie beim Lesen von Speicher)
  - der Stack-Pointer wird also wieder bis zum Format String geschoben

Format String Bug - Angriffe

- Überschreiben von Speicher (Forts.)
  - Zahl die in int* geschrieben wird, durch Anzahl Characters beeinflußbar
    - Bsp.: %15d
    - *Beispiel (8)*
  - dieser Einfluß ist aber nur gering
  - also "multiple writes on byte level" oder short – write (%hn – write to short), stackpopping, direct parameter access

Format String Bug - Angriffe

- Konsequenzen
  - Möglichkeit, jede denkbare Adresse mit selbst gewählten Daten zu überschreiben
  - also Return Adressen, GOT, function pointer, .DTOR, libc-memory-hooks,
- Erweiterte Techniken
  - return to libc (Solar Designer)

Format String Bug - Schutz

- Quellcode Review
  - Bsp: printf
    - nur der erste Parameter ist Format String
- libsafe
  - Austausch der betroffenen Funktionen zur Laufzeit
  - Überprüfen, ob die Funktion sicher ausgeführt werden kann
    - Return Adress and Frame pointer Check
    - Frame Span Check
  - wenn nicht Programmabbruch

Format String Bug - Schutz

- Non-exec-Stack, Canary zum Verhindern von eingeschleustem Code (ähnlich Buffer Overflows)

Gliederung

- Einleitung
- Schwachstellen
  - Buffer Overflows
    - stack
    - heap
  - Malloc – Probleme
  - Format String
- Zusammenfassung/Ausblick

Zusammenfassung

- Problembereiche
  - Eingaben in das Programm, denen nicht vertraut werden kann
  - Vermischung von Metadaten und Daten (Channeling Problem)
- Konsequenz
  - Eingaben nicht vertrauen
  - vermeiden von stdio ;)
  - Testen! Testen! Testen!

Channel Problem

Situation	Data Channel	Controlling Channel	Security Problem
Phone System	Voice or Data	Control Tones	seize line control
PPP	Transfer Data	PPP commands	traffic amplification
Stack	Stack data	Return Adress	control of retaddr
Malloc Buffers	Malloc Data	Managment info	write to memory
Format String	Output String	Format Parameters	format function control

Ausblick

- Programmierung nur ein Teil der Systementwicklung
- viele andere Schwachstellen:
  - C++ autopointer attacks
  - Race Conditions
  - Timing Attacks
  - Signals
  - File Permissions

Literatur

- "Smashing the Stack for Fun and Profit" (phrack.org)
- "Once upon a free" (phrack.org)
- "Exploiting Format String Vulnerabilities" (teso.org)
- "Secure Programming" (dwheeler.com)

Ressourcen

- Mailinglisten
  - bugtraq@securityfocus.com
  - full-disclosure@lists.netsys.com
- Webseiten
  - www.phrak.org
  - ww.lsd-pl.org
  - www.teso.org
  - www.cert.org
  - www.packetstorm.org

Ende

Diskussion