

# [TOOL] BofCheck – Buffer Overflow, Environment Variables Overflow and Format String Vulnerabilities Binary Tester

*Source:* <http://www.derkeiler.com/Mailing-Lists/Securiteam/2003-09/0052.html>

---

*From:* SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

*Date:* 09/16/03

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 16 Sep 2003 16:48:04 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

BofCheck – Buffer Overflow, Environment Variables Overflow and Format String Vulnerabilities Binary Tester

---

## DETAILS

BofCheck is a program that can test weak binaries for basic vulnerabilities. It can test for command line overflows, ENV overflows, and basic format string vulnerabilities. BofCheck utilizes ptrace() to analyze the stack during testing and report any overwritten stack addresses and other important data.

Tool source code:

```
/* BofCheck.c – Coded by sw @ .:[oc192.us]:. Security.
```

```
*
```

```
* Please email me (sw@oc192.us) with questions, comments,
```

```
* ideas for this as it is an ongoing project.
```

```
*
```

```
* Simple tool to test bins for stack overflows, still beta and
```

```
* probably buggy.
```

```
*
```

```
* Changes:
```

```
* This version utilizes the ptrace() method in order to delay
```

```

* the process and return valuable exploitation info in the
* event we do find a vulnerability.... *BSD will be supported soon..
*
* Objective:
* Checks command line arguments v.s. a simple/env/format strings overflow
* and returns the signal status, giving us information
* as to if the program is vulnerable or may be vulnerable
* to a simple command line overflow.
*/

```

```

#include <stdio.h>
#include <stdlib.h>
#include <sys/ptrace.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <asm/user.h>
#include <string.h>
#include <fcntl.h>
#include <signal.h>
#include <unistd.h>

```

```
#define LIST 16
```

```
/* create a list of the most common signal traps */
```

```

struct sigs{
    char *type;
    char *num;
} types[] = {
    {"SIGINT", "2"},
    {"SIGQUIT", "3"},
    {"SIGILL", "4"},
    {"SIGTRAP", "5"},
    {"SIGABRT", "6"},
    {"SIGEMT", "7"},
    {"SIGFPE", "8"},
    {"SIGKILL", "9"},
    {"SIGBUS", "10"},
    {"SIGSEGV", "11"},
    {"SIGSYS", "12"},
    {"SIGPIPE", "13"},
    {"SIGALRM", "14"},
    {"SIGTERM", "15"},
    {"SIGUSR1", "16"},
    {"SIGUSR2", "17"},
};

```

```

char args[1000], a[100], *prs;
int verbose=0, strings=0, argmode=0, nigsiz=10000, envmode=0;
/* logging function */
void log(char *program, char *args, char *sig, char *jizz, char *sp, char
*bufaddr, char *logfile){

```

```
FILE *logz;
logz=fopen(logfile, "a");
fprintf(logz, "[%s] with option %s \nCAUGHT[%s]Signal[%s]\n",
        program, args, jizz, sig);
fprintf(logz, "* [ESP] at time of crash [%s]\n" , sp);
fprintf(logz, "* [0x41414141] string possibly overwrote address[%s]\n",
bufaddr);
fprintf(logz, "\n");
fclose(logz);
}
```

```
/* usage function */
void usage(char *yourself){
    fprintf(stderr,
        "oc192-bof.c – coded by sw @ .:[oc192.us]:. Security\n"
        "Usage: %s -f <file> [options]\n"
        "\n"
        "Options:\n"
        "-s: Use this to test for generic format strings bugs\n"
        "-e: Specify ENV variables to try an overflow\n"
        "-b: Set cmd line buffer overflow size (Default: 10000)\n"
        "-a: Specify arguments to test (Default: A-Z,a-z)\n"
        "-l: Specify logfile (Default: bofcheck.log)\n"
        "-h: Help/examples\n"
        "-v: Verbose mode\n"
        "\n", yourself);
    exit(0);
}
```

```
void help(char *yourself){
    fprintf(stderr,
        "Examples:\n"
        "\n"
        "-----\n"
        "NOTE: -e,-a,-s cannot be used in combination.\n"
        "\n"
        "-----\n"
        "%s -f <file>:\n"
        "Performs cmd line overflow check on all args A-Z, a-z\n"
        "\n"
        "-----\n"
        "%s -sf <file>:\n"
        "Performs generic format string test on all args A-Z, a-z\n"
        "\n"
        "-----\n"
        "%s -a -a,-b,-c -f <file>:\n"
        "Performs cmd line overflow check on args -a -b -c\n"
        "\n"
        "-----\n"
        "%s -e TERM,SIZE -b 6000 -f <file>:\n"
        "Attempts to overflow TERM and SIZE env variables on selected
```

```

file\n"
"
_____\n"
    "%s -s -a -a,-b,-c -f <file>:\n"
    "Performs generic format string test on args -a -b -c\n"
"
_____\n"
    "%s -b 2038 -vf <file>:\n"
    "Performs generic format string test on all args A-Z, a-z
with\n"
    "buffer of 2038 and verbose output\n"
"
_____\n",
    yourself, yourself, yourself, yourself, yourself, yourself);
    exit(0);
}

/* ptrace routine to analyze target programs' regs and exit status */
int pnic(char *path, char *program, char *args, char *buf, char *logfile){
unsigned long aa;
struct user_regs_struct regs;
int pid_vuln, z, status, i, fd;
char sp[50], bufaddr[50];

    /* fork victim program into memory */
    if (!(pid_vuln = fork())){
    fd = open("/dev/null", O_WRONLY|O_CREAT|O_TRUNC, S_IRWXU);
    dup2(fd, STDERR_FILENO); /* Kill messy output from child */
    dup2(fd, STDOUT_FILENO);
    close(fd);
    alarm(5);
    if(envmode==0){
    execl(path, program, args, buf, NULL);

    printf("/* Failed: execl %s/%s\n", path, program);
    exit -1;
    }
    if(envmode){
    setenv(args, buf, 1);
    execl(path, program, NULL);
    }
    }
    /* attach to the child */
    if (ptrace(PTRACE_ATTACH, pid_vuln)){
    printf("/* Failed: PTRACE_ATTACH\n");
    return -1;
    }
    }
    waitpid(pid_vuln, NULL, 0);

```

```

/* allow the program to continue running */
if (ptrace(PTRACE_CONT, pid_vuln, 0, 0)){

    printf("** Failed: PTRACE_CONT\n");
    exit(-1);
}
waitpid(pid_vuln, NULL, 0);

/* grab programs registers */

if (ptrace(PTRACE_GETREGS, pid_vuln, 0, &regs)){
    printf("** Failed: PTRACE_GETREGS\n");
    return -1;
}

/* store the SP for the log() */
sprintf(sp, "0x%08x", (int) regs.esp);
z = 0, aa = 0;
/* look at regs to check if any addresses have been overwritten by our
test overflow */

do{
if ((aa = ptrace(PTRACE_PEEKTEXT, pid_vuln,
(int)(regs.esp+(z++)), 0)) == -1)
{
printf("** Failed: PTRACE_PEEKTEXT.\n");
return 1;
}
} while (aa != 0x41414141);
z--;
sprintf(bufaddr, "0x%08x", (int)(regs.esp + z));

/* release program from ptrace() since we have mooched all our info */
if(ptrace(PTRACE_DETACH, pid_vuln, 0, 0)){
printf("** Failed: PTRACE_DETACH\n");
exit(-1);
}
else {
/* look at exit status and log accordingly */
usleep(50000);
if((pid_vuln = waitpid(-1, &status, WNOHANG )) == -1 ) {
printf( "** FAILED: Wait error\n" );
}
if(WIFEXITED(status) != 0 ) {
if(verbose){
printf("Normal exit.\n");
}
}
}
else{
sprintf(a, "%d", status);
if(strlen(a) > 4){
exit(0);
}
}
}

```

```

    }
    for (i=0;i<LIST;i++) {
        if(strstr(a, types[i].num)){
            printf("Abnormal exit with %s, SIG[%d][%s]\n", args, status,
                types[i].type);
            log(program, args, a, types[i].type, sp, bufaddr, logfile);
        }
    }
    }
    waitpid(pid_vuln, NULL, 0);
}
}
int main(int argc, char *argv[]){
int i, c;
char *file, *arg, *logfile, *env;
/* duh */
if(argc < 2){
    usage(argv[0]);
    exit(0);
}
logfile = "bofcheck.log";
while((c = getopt(argc, argv, "f:a:e:b:l:svh"))!= EOF){
switch(c){
    case 'f':
        file = optarg;
        break;
    case 'a':
        argmode++;
        arg = strdup(optarg);
        break;
    case 'e':
        envmode++;
        env = strdup(optarg);
        break;
    case 'b':
        nigsawsize = atoi(optarg);
        if(nigsawsize==0){
            printf("** Bad buffer value !. Exiting\n");

            exit(-1);
        }
        if(nigsawsize > 99999){
            printf(" * Buffer size too large ! Try <99999. Exiting\n");
            exit(-1);
        }
        if(nigsawsize < 100){
            printf(" * Buffer size too small ! Try 100<. Exiting\n");
            exit(-1);
        }
        break;
    case 'l':

```

```

        logfile = optarg;
        break;
    case 'h':
        help(argv[0]);
        return 1;
    case 's':
        strings++;
        break;
    case 'v':
        verbose++;
        break;
    default:
        usage(argv[0]);
        return 1;
    }
}
if(open(file, O_EXCL|O_RDONLY) == -1){
    printf("* Cant open file ! Please specify a valid filename.
Exiting\n");
    exit(-1);
}
if(strlen(file) > 10000){
    printf("* File name too long !. Exiting\n");
    exit(-1);
}
if(envmode){
    env_test(file, env, logfile);
    exit(0);
}
if(strings){
    fmt_test(file, arg, logfile);
    exit(0);
}else{
    test(file, arg, logfile);
    exit(0);
}
}

/* test file with overflow on args */
int test(char *file, char *arg, char *logfile){
    int i;
    char path[10000], program[10000], buf[nigsize];
    bzero(buf, sizeof(buf));
    memset(buf, 0x41, nigsize);
    sprintf(path, "%s", file);
    sprintf(program, "%s", file);
    if(argmode){
        printf("[+]Testing %s args %s\n", program, arg);
        if((prs = strtok(arg, ",")) == NULL){
            printf("* Bad string pussy !. Exiting.\n");
        }
    }
}

```

```

    if(verbose){
        printf("%s: argv %s: buffer[%i]..", program, prs, nigsaw);
    }
    pnig(path, program, prs, buf, logfile);
    while ((prs = strtok(NULL, ",")) != NULL){
        if(verbose){
            printf("%s: argv %s: buffer[%i]..", program, prs, nigsaw);
        }
        pnig(path, program, prs, buf, logfile);
    }
    exit(0);
}
printf("[+]Testing %s args A-Z,a-z\n", program);
for(i='A'; i<='z'; i++) {
    sprintf(args, "-%c", i);
    if(verbose){
        printf("%s: arg %s: buffer[%i]..", program, args, nigsaw);
    }
    pnig(path, program, args, buf, logfile);
}
}
int env_test(char *file, char *env, char *logfile){
char path[10000], program[10000], buf[nigsaw];
bzero(buf, sizeof(buf));
memset(buf, 0x41, nigsaw);
sprintf(path, "%s", file);
sprintf(program, "%s", file);
if(argmode){
    printf("* -a cannot be used with -e ! Exiting\n");
    exit(0);
}
if(strings){
    printf("* -s cannot be used with -e ! Exiting\n");
    exit(0);
}
printf("[+]Testing %s with ENV vars %s\n", program, env);
if((prs = strtok(env, ",")) == NULL){
    printf("* Bad string pussy !. Exiting.\n");
}
if(verbose){
    printf("%s: ENV var %s: buffer[%i]..", program, env, nigsaw);
}
pnig(path, program, prs, buf, logfile);
while ((prs = strtok(NULL, ",")) != NULL){

if(verbose){

    printf("%s: ENV var %s: buffer[%i]..", program, prs, nigsaw);
}
    pnig(path, program, prs, buf, logfile);
}
}

```

```
    }
}
int fmt_test(char *file, char *arg, char *logfile){
int i;
char path[10000], program[10000], string[50];
bzero(string, sizeof(string));
strcat(string, "%s%n%d%c%i%s%n%d%c%i");
for(i=0; i < 9; i++)
    strcat(string, "\\x41");
sprintf(path, "%s", file);
sprintf(program, "%s", file);
if(argmode){
printf("[+]Testing %s args %s with fmt_strings\n", program, arg);
if((prs = strtok(arg, ",")) == NULL){
printf("* Bad string pussy !. Exiting.\n");
}
if(verbose){
printf("%s: argv %s: fmt_string..", program, prs);
}
pnig(path, program, prs, string, logfile);
while ((prs = strtok(NULL, ",")) != NULL){
if(verbose){

printf("%s: argv %s: fmt_string..", program, prs);
}
pnig(path, program, prs, string, logfile);
}
exit(0);
}
printf("[+]Testing %s args A-Z,a-z with fmt_strings\n", program);
for(i='A'; i<='z'; i++){
sprintf(args, "-%c", i);
if(verbose){
printf("%s: arg %s: fmt_string..", program, args);
}
pnig(path, program, args, string, logfile);
}
}
```

#### ADDITIONAL INFORMATION

The information has been provided by <<mailto:sw@oc192.us>> sw.

=====

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

[list-unsubscribe@securiteam.com](mailto:list-unsubscribe@securiteam.com)

In order to subscribe to the mailing list, simply forward this email to: [list-subscribe@securiteam.com](mailto:list-subscribe@securiteam.com)

=====  
=====

**DISCLAIMER:**

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.