

Bug Wars

[The Lost Matrix of Security Vectors]

[General Research]

Aditya K Sood aka 0Kn0ck
Sec Niche Security
www.secniche.org

[Bug Wars : The Lost Matrix of Security Vectors]

Contents

- 1] Abstract .
- 2] Anatomy of Bug.
- 3] Realm of Exploitation.
- 4] Panorama of Security Vectors.
- 5] The Identity.
- 6] The Weapons of Bug War.
 - 6.1] The Black Pointers.
 - 6.2] Division By Black Zero.
 - 6.3] Overflow-Underflow Black Crux.
 - 6.4] Black Memory Access Violations .
 - 6.5] The Black Liquid : Memory Leakage.
 - 6.7] Dead Limits : Buffer Overflows.
- 7] Concern : Bug Origination.

The Inference.

[1]Abstract

The Matrix War!

Exploitation is all about taming the undefined elements into required parametric layout, when these parameter changes, the control is transferred and the system is nothing but a slave of brilliant minds. The bug is the triggering element in this. This paper describes the stringent anatomy of bugs and security vectors from a general perspective. The point of this talk is to understand the pros and cons of bug origination keeping in mind the technology ailment. A conceptual layout.

According to a saying in The Cathedral and The Bazaar By Eric Raymond.

"Given Enough Eyeballs all bugs are shallow"

Lets walk along it.

[2] ***The Anatomy of Bug!***

The bug persists in the matrix of computer system. These bugs have implicit ramifications on the system if security consideration is undertaken. Have we ever thought over the criticality of bug wars, the cause? Do we really want these bug wars to be ended or go on continuously? What will happen if bug war ends? One thing is sure it definitely results in matrix stagnation. The bugs have their own relative significance. On the contrary these bugs are considered to be as base for ongoing software development and security.

[2.1]The technology world is surging ahead with enormous speed and it becomes hard to put constraint. The bug present in the system emanate from our own faults. The bugs have both positive and negative implications. Let's look at the positive aspect. The dooming of these bugs has positive reflection as these insecure objects trigger development and innovation.

[2.2]New innovative methods are designed to remove the overhead created by self activated bugs. Why should not we look at the negative side too? Of course the paradigm revolves around loss of system resources, disruption in network, insecurity in programs etc. The bugs have dependencies that are cross structured. The crux is to understand the standard definition of bug relativity.

It is imperative to set the thinking vector in the context of bug specification because the arrow of development initiates from there. The occurrence of bug is crucial in the system. The protection and exploitation realm are interdependent. Both aspects are defined over generation of bugs.

[3] ***The Realm of Exploitation.***

Exploitation of vulnerability is the key to get the work done. This enables one to command the machine and outmaneuver its working. The attack and defense are the two sides of a same coin that encapsulate themselves in the arena of security and hacking.

[3.1]Security is a dominant factor in the world of computers. It encompasses not only the methods and mechanisms to secure the machines and networks but also the obscurity behind exploitation. It determines which parameter of security becomes the weakest link for the penetration. This is one of the prime factors of security and serves as foundation to reverse the technique and learning the mechanisms of technology.

[3.2]The paradigm of security depends on some specific vectors that pave the way to determine the securities or insecurities intrinsic in the system. Their magnitude describes the level of security applied and rigorous effects on the network and machines hosting that security vectors. Why Security vectors ? The vector in itself is prominent as it sets the magnitude and the direction for the environment to work efficiently.

[4] ***Panorama of Security Vectors***

The security vectors are more structural in their working approach. This layout explains security in terms of very definite vectors which will be applied mathematically to the realm of security to carve out the new persistent standards. The security itself delves into a new paradigm incessantly as new technology is evolving along with the parameters. In order to understand this parametric layout a detailed analysis is indispensable because core knowledge strengthens the hold on security parameters.

[4.1]The exploiting parameters are also helpful in grappling with the technology more efficiently as these parameters define the weaknesses persisting in the security domain and how it traverses out to the main stream thereby jeopardizing the whole specific domain. So all these relevant entities encompass the security panorama.

[4.2]There are myriads of security models,some of them become the benchmarks but some are adapted to comply the present yardsticks of security. Ignoring this will instigate the exploitation parameters to wreck the networks The human learning is indispensable in the practical implementation of security to grapple with the transformations in the technology. Both the factors must be diffused in a definitive manner to dethrone the insecurities and instabilities inherent in the security domain. This works in both the ways as pros and cons are intrinsic to the technology making the technology realm very subtle.

[4.3]The pace of development is going on not only result in the outbreak of growth but also cause the demise of security domain if not handled in the right sequential manner. We will look at the security vector in detail, how to implement it in the mathematical way to understand the security domain more clearly.

The stress is on the security parameters that define the domain of security upon which the security vectors are defined. We are going to analyze the every single entity that has implications on the security domain. The Security Domain is defined as the set of parameters that affect the security. We define security domain as

$$Sd = \{ Authentication , Authorization , Confidentiality , Integrity , Access Control \}$$

The $S(t)$ is called to be as the Security vector that relates to every single security parameter that is defined in the Security Domain(Sd).

$S(t)$ => *The Security Vector Function.*
 S => *The Security Vector*
 t => *The Stress Limit Of Security Parameters.*

Integrating all of the parameters based on the domain parameters results in:

$$S(t) = S1(t)Confidentiality + S2(t)Integrity + S3(t)Authentication + S4(t)Authorization + S5(t)Access Control$$

So if we analyze this mathematical layout we can simply conclude that all the security parameters must have stress limit. Lets look at the continuity of security vectors in the context of bug wars. The continuity plays a crucial role in the security implications because for the security to be stringent it should be continuous and impregnable. Let's look at the continuity of security vector. The basic considerations are:

*=> S(t) parameter check should be defined.
=> If S(t) is clearly applicable for any type of stress limit t.*

*Lets undertake security entity (a) with stress limit time t1
Lets undertake security entity (b) with stress limit time t2*

The security vector is continuous if

$$S(a) \text{ at } t1 = S(b) \text{ at } t2$$

This defines the security entity in equilibrium because after the time phases out the security vector is constant. This means the security parameters become hard enough to dethrone the encountered flaws. We will look at the CIA Model called as Confidentiality, Integrity And Availability Model.

The attack space is termed as the space in the security domain where the attacks are possible, overriding the present security constraints. Thus the attack space marginalizes the security domain. Let's look into it:

S(t) Security Vector function defined.

*S1(t) at t=t1 with stress limit t1 after which attack occur in S1
S2(t) at t=t2 with stress limit t2 after which attack occur in S2
S3(t) at t=t3 with stress limit t3 after which attack occur in S3
So with full domain parametrics:*

$$S(t) = S1(t) \text{ at } t=t1(i) + S2(t) \text{ at } t=t2(j) + S3(t) \text{ at } t=t3(k)$$

You can look very clearly i , j , k space directions are defined. We set our base on the CIA model for attack space analysis:

i => Confidentiality , j => Integrity , k => Availability

So three dimensional security considerations are undertaken. The Attack space is negligible provided the vector function is continuous at every time phase This does not allows the attack parameters to have their space in the security domain and cause the attack. So S(t) is considered to be as continuous if and only if:

S1(t) at t=t1 , S1(t) at t=t2 , S1(t) at t=t3 are continuous.

This holds that for whole security vector to be continuous, then the individual security vectors have to be continuous too. If this is true the attack space tends to zero and the attack parameters are dethroned.

[5] ***The Identity!***

The bugs possess their own identity and are classified on the basis of their mode of penetration. It relates to the manner they play with the system.

[5.1]The importance of identity of these bugs can be gauged from the fact that the development vector is based on it. The war in general always occurs to prove the better of the one out of two metals. This is because the one who possesses authority rules the system. Still the devastation result from one element but the growth and development from another.

[5.2]The war holds uniqueness in its own context because it subjugates the new era of software and provides more security and protection to the upcoming generation with more suitable and desired working entities. The bug war holds the key of hidden traits of system which should be exposed to see what lies beneath in the matrix.

[5.3]The bug war layer by layer removes the surface of hidden artifacts. It is getting very interesting to look into the specifications of bug and their relative profound impacts on the systems.

The above layout shows the structure of various bugs and the black impact those bugs have on the system. Ever since a bug initiates the result is always negative if seen primarily from system context. The bugs are considered to be as weapons of system exploitation.

[6] ***The Weapons of Bug War***

According to tradition every single war requires weapon. So what the bug war actually requires. Without the triggering element in war the fight between two objects are not possible. So from technology point of view what are the weapons that are used to raise fire between exploitation and security. Lets look at some of the weapons:

[6.1] ***Weapon : The Black Pointers***

The pointers are considered to be as foremost exploitation weapon from security point of view. The pointers allow program to access objects that are not explicitly declared. Look at the scenario of null pointers which refer to no object and not even considered as zero valued. The insecure vector persists when ever a dereference operation is used against NULL vector. The result is unhandled exception and termination of program matrix.

Generally termed to be as Null Pointer Exception. Uninitialized pointers have different stats than NULL pointers. Malloc implementation in C with Errno dependency not necessarily prevents the exception. The uninitialized pointers are considered as Wild Pointers because it directs to unhandled addresses there by causing program crash. The double redirection layout of pointers are dangerous too. Certain cases:

6.1.1] when ever a pointer try to write beyond the end of buffer , stack is destroyed.

6.1.2] Accessing outside data structures through pointers.

6.1.3] Double Indirection of pointers.

6.1.4] Using pointer to heap when memory is already de-referenced.

So above factors intensify the element of matrix infection through pointers. The bug war flourish through black pointers. No doubt in that.

A simple structural code of Heap memory dereferencing:

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

int main(void)
{
    /* initialize pointer to heap */
    int *p1 = malloc(sizeof *p1);

    /* make a copy */
    int *p2 = p1;

    /* initialize the heap */
    *p1 = 0;

    /* points into the heap */
    printf("Address of p2: %p\n", (void *)p2);

    /* should print zero */
    printf("Value of *p2: %d\n", *p2);

    /* deallocate the memory */
    free(p1);

    /* other code, possibly using the heap */
    /* p2 still points to the original allocation,
     * but who knows what is there */
    printf("Value of *p2: %d\n", *p2);

    /* invalid use of p2 */

    return 0;
}
```

Ofcourse looking at this its hard to handle the exception that will generate due to dereferencing of memory pointer.

[6.2] Weapon : Division By Black Zero

Even a zero cause havoc in system .so must be termed as "Black Zero". Mostly I have seen people think what a zero can do. Looking into the system perspective Zero can do that thing which is very hard to think. This anonymity can be considered as inbuilt or application prone but result occurs in unexceptional handling of bug. The "Division by zero" bug. The contradiction persist between division of zero in integers and floating point numbers. The CPU handle the both operations differently. The processors generates an exception bug continuously or may be false results can be displayed. Even the pseudo inverse fails in complying with this bug.

A bit of Mathematical journey:

At first glance it seems possible to define $\frac{a}{0}$ by considering the limit of $\frac{a}{b}$ as b approaches 0.

For any positive a , it is known that

$\lim_{b \rightarrow 0^+} \frac{a}{b} = +\infty$
and for any negative a ,

$$\lim_{b \rightarrow 0^+} \frac{a}{b} = -\infty.$$

Therefore, considering $\frac{a}{0}$ as $+\infty$ for positive a , and $-\infty$ for negative a . The inconveniences are :

[6.2.1] Positive and negative infinity are not real numbers. So as long as we wish to remain in the context of real numbers, we have not defined anything meaningful.

[6.2.2] Taking the limit from the right is arbitrary. We could just as well have taken limits from the left and defined $\frac{a}{0}$ to be $-\infty$ for positive a , and $+\infty$ for negative a . This can be further illustrated using the equation (assuming that several natural properties of reals extend to infinities)

$$+\infty = \frac{1}{0} = \frac{1}{-0} = -\frac{1}{0} = -\infty$$

which does not make much sense. This means that the only workable extension is introducing an unsigned infinity. Furthermore, there is no obvious definition of $\frac{0}{0}$ that can be derived from considering the limit of a ratio. The limit

$$\lim_{(a,b) \rightarrow (0,0)} \frac{a}{b}$$

does not exist. Limits of the form

$$\lim_{x \rightarrow 0} \frac{f(x)}{g(x)}$$

in which both $f(x)$ and $g(x)$ approach 0 as x approaches 0, may converge to any value or may not converge at all. Based on L Hospital Rule. Even this particular approach cannot lead us to a useful definition of $\frac{0}{0}$.

The ambiguity persists in the implementation of this bug which hampers the functioning of program matrix.

[6.3] Weapon : Overflow / Underflow Black Crux

It is also considered to be as a classic infection weapon considering system bugs. When ever a rogue condition produces a result that is not handled by the pre defined register , the black overflow occurs. Looking at arithmetic perspective when ever two operands undergo any operation and produces a result with no specific sign relative to the operand signs , then the exact overflow occurs. Like when two signed integers are operated and result comes out to be unsigned due to any reason the registers loose there integrity there by rendered weak in front of overflow stature. A carry element in operations act as a rogue point to initiate overflows. Lets turn on the other side i.e. underflows. The cause is overflow of the exponent quantity of floating point numbers. so the underflows and overflows delve into the same pool with different swimming strokes. so lets have a look on black bugs of overflow:

[Openssh 3.3 Integer Overflow]

```
nresp = packet_get_int();  
  
if (nresp > 0)  
{  
    response = xmalloc(nresp*sizeof(char*));  
    for (i = 0; i < nresp; i++)  
        response[i] = packet_get_string(NULL);  
}
```

A better look at :

```
char* processNext(char* strm)  
{  
    char buf[512];  
    short len = *(short*) strm;  
    strm += sizeof(len);  
    if (len <= 512)  
    {  
        memcpy(buf, strm, len);  
        process(buf);  
        return strm + len;  
    }  
    else  
    {  
        return -1;  
    }  
}
```

The bug triggers up when ever the bounded limit is crossed as per defined examples. The black friday will occur for the Open ssh . Mostly a programming error but bug war will start definitely. The overflow and underflow entities are considered to be as two sides of same coin bu the only difference is the mode of exploitation and exception generation.

[6.4] Weapon : Black Memory Access Violations

Some computer programs are subtle and try to intrude into the matrix of memory objects that are not defined for them. But they still do. As a result access violations occur. The system adheres to instability as matrix is going to be crashed due to these unhandled operations. But a question arises why the program try to access the undefined addresses in matrix. Oh! my god the rogue conditions push the program directive to point towards the curved matrix. The processors termed these conditions as Bus Errors in a relation to hardware used. The dependencies change with platform used.

```
const char *temp = "Hackers U There?";
*s = 'God'

$ gcc segfault.c -g -o segfault
$ ./segfault
Segmentation fault

Program received signal SIGSEGV, Segmentation fault.
0x1c0005c2 in main () at segfault.c:6
*temp = 'God';

$ gcc segfault.c -g -o segfault
segfault.c: In function 'main':
segfault.c:4: error: assignment of read-only location
```

Lets taste windows:

```
Faulting application DrvInst.exe, version 6.0.6000.16386, time stamp 0x4549ad51,
faulting module ntdll.dll, version 6.0.6000.16386, time stamp 0x4549bdc9,
exception code 0xc0000374, fault offset 0x000af1c9, process id 0x158c,
application start time 0x01c7cc85828a1e8a.
```

```
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
  <System>
    <Provider Name="Application Error" />
    <EventID Qualifiers="0">1000</EventID>
    <Level>2</Level>
    <Task>100</Task>
    <Keywords>0x8000000000000000</Keywords>
    <TimeCreated SystemTime="2007-07-22T17:27:33.000Z" />
    <EventRecordID>15354</EventRecordID>
    <Channel>Application</Channel>
    <Computer>FamilyComp</Computer>
    <Security />
  </System>
```

Another one:

The instruction at 0x00366f9d referenced memory at 0x00000f18. The memory could not be read.

The above stated layout depicts the memory access violation occur due to program errors. The memory could not be read by the object and as predicted system shows segmentation fault.

[6.5] Weapon : The Black Liquid : Memory Leakage

The memory leakage holds complexity in its own term. The curvature sets to dark when ever a computer program consuming memory but unable to release it even after the working functionality is over. The bug starts to flourish through memory leakage. It is termed as Black Liquid because the memory leakage bug going to affect the stature of application in a weird manner. A program even loses the ability to access the memory efficiently there by generating disrupting element in system. The bug war initializes through this act. The exploitation vector dethrones the robust functionality of processing system by illicit consumption of memory. The subtle pointers when not applied properly in an application layout dereferences to memory leakage.

Lets have a look : IE Memory Leakage.

```
<html>
<head>
  <script type="text/javascript">
    function LeakMemory(){
      var parentDiv =
        document.createElement("<div onclick=foo()>");
        parentDiv.bigString = new Array(1000).join(
          new Array(2000).join("XXXXX"));}
    </script>
  </head>
  <body>
    <input type="button"
      value="Memory Leaking Insert" onclick="LeakMemory()" />
  </body>
</html>

<script type="text/javascript">
  var myGlobalObject;
  function SetupLeak(){
    myGlobalObject=document.getElementById("LeakedDiv");
    //The memory will leak if not handled properly.
    document.getElementById("LeakedDiv").expandoProperty=
      myGlobalObject;
  }
</script>
</head>
  <body onload="SetupLeak()">
  <div id="LeakedDiv"></div>
```

The above stated layout describes a memory leakage flaw in internet explorer. The bug act as a parasite on system processing.

A Specific C layout:

```
#include <stdio.h>
#include <stdlib.h>
void f_leak(void)
{
    void* s_leak;
    s_leak = malloc(50); /* get memory */
    return; /* memory leak - see note below */

    * Memory was available and pointed to by s_leak, but not saved.
    * After this function returns, the pointer is destroyed,

}

int main(void)
{
    /* this is an infinite loop calling the above function */
    while (1) f_leak();
    /* Malloc will return NULL sooner or later, due to lack of memory */
    return 0;
}
```

So a rigorous war can be started in a system due this black liquid drained in the system processing pool. A truth :

" If there is no memory leak in kernel , the OS will fail "

This is complex. Really complex.

[6.6]Weapon : Dead Limits – Buffer Overflows

One of the major cause of system wars. Its an memory access violation due to programming error. The eip register sets in promiscuous mode after buffer overflow bug. The exploitation is fruitful only when the eip bug directs to functional part of registers. It means the bug is exploited when eip is controlled by the brilliant minds. This can change the bug war strategy and even the consequences related to system arena. The matrix is manipulated at core. The bug directs toward function overwriting , overriding return values and local variables. These three standard vectors results in generation of buffer overflow bugs. Stack overflows , Heap overflows are incore part of buffer exploitation. The exploitation is performed by tempering internal data structures like stack or heap. Strangled buffer without limits cause havoc to program application in weird stringent manner.

A General Layout :

```

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

int main(int argc, char *argv[])
{
    char buffer[10];
    if (argc < 2)
    {
        fprintf(stderr, "USAGE: %s string\n", argv[0]);
        return 1;
    }
    strcpy(buffer, argv[1]);
    return 0;
}

```

A Heap overflow:

```

int main(int argc, char **argv)
{
    HANDLE h = NULL;
    LPVOID mem1 = NULL, mem2 = NULL, mem3 = NULL;
    unsigned char shellcode[128];
    if (getaddr() != 0)
        return 0;
    h = HeapCreate(0, 0, 0);
    if (h == NULL) { printf("Error: HeapCreate failed\n");
        return 0; }
    printf("Heap: %08X\n", h); mem1 = HeapAlloc(h, 0, 64-8);
    printf("Heap block 1: %08X\n", mem1); mem2 = HeapAlloc(h, 0, 128-8);
    printf("Heap block 2: %08X\n", mem2);
    HeapFree(h, 0, mem1); HeapFree(h, 0, mem2);
    mem1 = HeapAlloc(h, 0, 64-8); printf("Heap block 1: %08X\n", mem1);

    // buffer overflow occurs here...
    memset(mem1, 0x31, 64);
    memcpy((char *)mem1+64, "\x84\xff\x12\x00", 4);
    mem2 = HeapAlloc(h, 0, 128-8); printf("Heap block 2: %08X\n", mem2);

    mem3 = HeapAlloc(h, 0, 128-8); printf("Heap block 3: %08X\n", mem3);
    memset(shellcode, 0, sizeof(shellcode)-1);
    // fake ret address
    memcpy(shellcode, "\x8B\xff\x12\x00", 4);
    // shellcode - "calc.exe"
    memcpy(shellcode+4, "\x90\x90\x90\x90", 4);
    memcpy(shellcode+4+4, calc_code, sizeof(calc_code)-1);
    // overwrite stack frame
    memcpy(mem3, shellcode, sizeof(calc_code)-1+8);
    return 0; }

```

Its very hard to combat against bug wars comprising of heap and stack incore weapons.

[7] Concern : Bug Origination

Look at the causes of the origination of bugs. The prime cause is programming errors that cause the bugs to trigger in the system thereby throwing uncontrollable exceptions. The exploitation theory states that the exceptions generate the loopholes that infiltrate the system matrix. The variance in vector occurs that is out of the bounds of matrix limit. As the variance is not handled in the defined parameters, so the vector points to some undefined space. This space provides the base for exploitation. It is all about taming the undefined elements into the required parametric layout, when these parameter changes happen; the system is nothing but a slave of brilliant minds. The bug is a initiating factor in this. As the matrix flourishes with time , bugs are transformed from the traditional to modern objects. The internal structure is transformed, so it is quite apparent that the vectors and parameters are going to be changed. The windows throw Blue Screen Of Death, programmers worst dream and Linux kernel panic is some of the new era exploit bases. The Y2K bug and Pentium FD1V bugs are also new class but the implications are very hard on the desired systems. The impact is ruinous if not controlled properly.

The Inference!

The bugs are versatile and to some extent are not even classified as confirmed or unconfirmed bugs. Why this entity persists in the bug war strategy? This is because some bugs attack from back i.e. backstabbing the computer systems, making the tracking process an arduous task .The factor of space sometimes becomes one of the hardest elements to be defined because the computer matrix is so complicated, it is not easy to traverse the whole structure in one flow. It is very crucial to find those hidden space parameters that serve the penetration base for the matrix exploitation. The protection and exploitation are exchanging hands so fast, that this critical point has been ignored. As a result bug wars are proliferating like encaustic fire. So, dilemma is whether to control these bugs or let the war to go on. As an author my personal thought is that the occurrence of bug is also a crucial factor in development but the limits have to be defined. We must have great working strategy for security and protection. The mechanisms have to be so profound so that the versatile factor of harm should be abridged. This looks strange but very practical if we look at the present scenario of security industry. As soon as new entity developed the bypassing measures are too, so why the clarity is still out of the way regarding the bug panorama. At the end I would like to say

The Bug wars will never end.