2017 中国互联网安全大会
China Internet Security Conference

起源游戏引擎的漏洞挖掘与利用

郑吉宏 & Amat

长亭科技
安全研究员

自我介绍

Amat Cama

- Senior Security Researcher at Chaitin Tech
- CTF player, team Shellphish
- Pwning

郑吉宏

- 长亭安全研究员
- CTF 选手, Tea Deliverer 成员
- Pwning

北京长亭科技

- chaitin.cn
- 专注于为企业级用户提供下一代应用层安全解决方案
- 雷池（SafeLine） - 下一代web应用防火墙
- 谛听（D-Sensor） - 内网威胁感知系统
- 长亭安全研究实验室
  - 2017 Pwn2Own 第三
  - 2017 BlackHat 分享“一石多鸟：利用单个SQLite漏洞破解多个软件”
  - 2017 DEF CON CTF Tea Deliverer 战队获得第五

为何研究游戏引擎

·游戏用户基数巨大

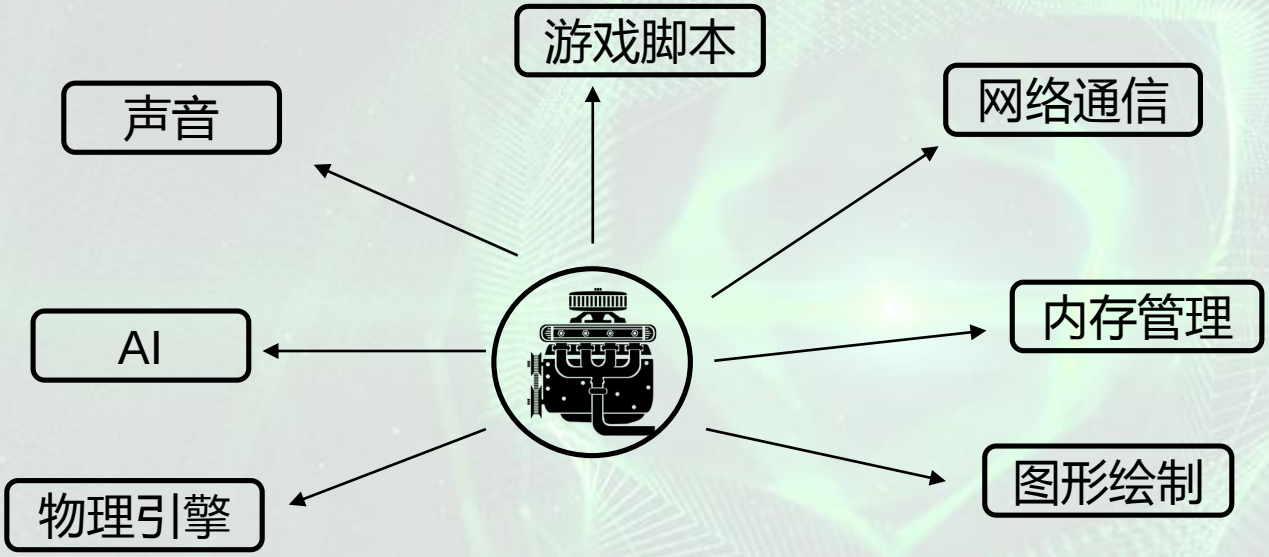
·游戏公司普遍不够重视安全

·JUST FOR FUN

何为游戏引擎

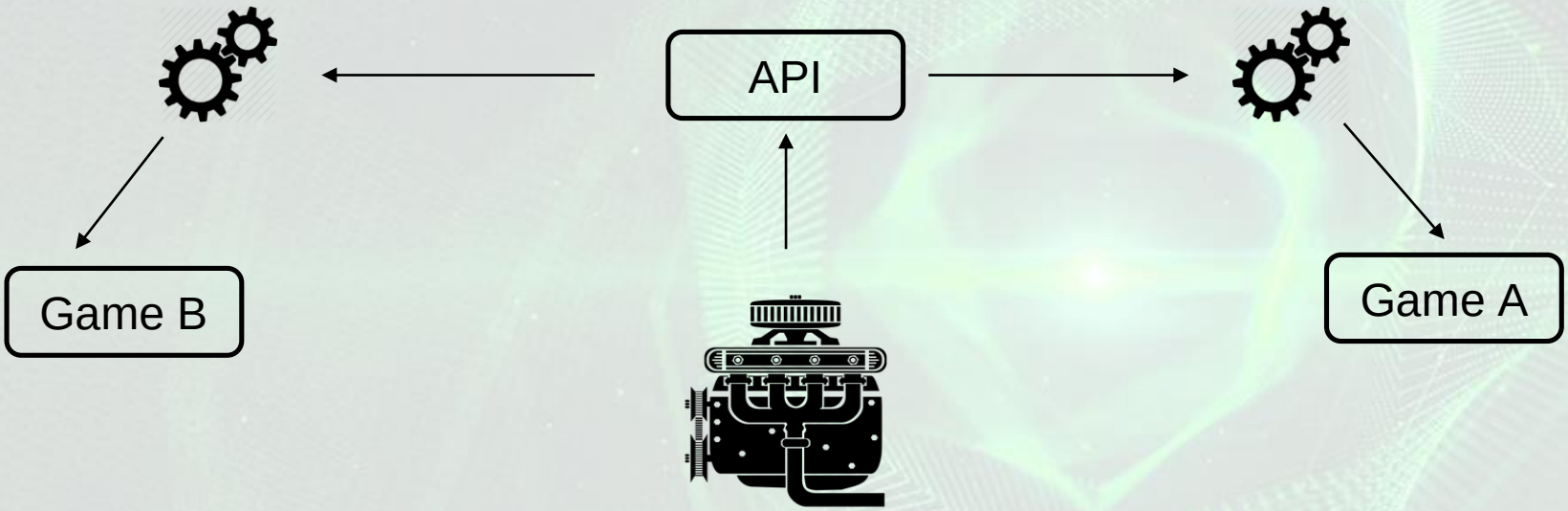
何为游戏引擎

- 游戏引擎相当于游戏的内核，是一个游戏的核心组件
- 提供了现成的渲染引擎、物理引擎、碰撞检测系统、音效、脚本引擎、电脑动画、人工智能、网络引擎以及场景管理等等功能
- 开发者可以利用现成的游戏引擎快速的进行游戏开发



何为游戏引擎

- 游戏引擎提供了实现不同功能的API接口
- 开发者可以使用官方提供的SDK来调用接口，实现高度可定制化的游戏开发



常见的游戏引擎

·寒霜引擎（FROSTBITE）：战地系列，FIFA系列…

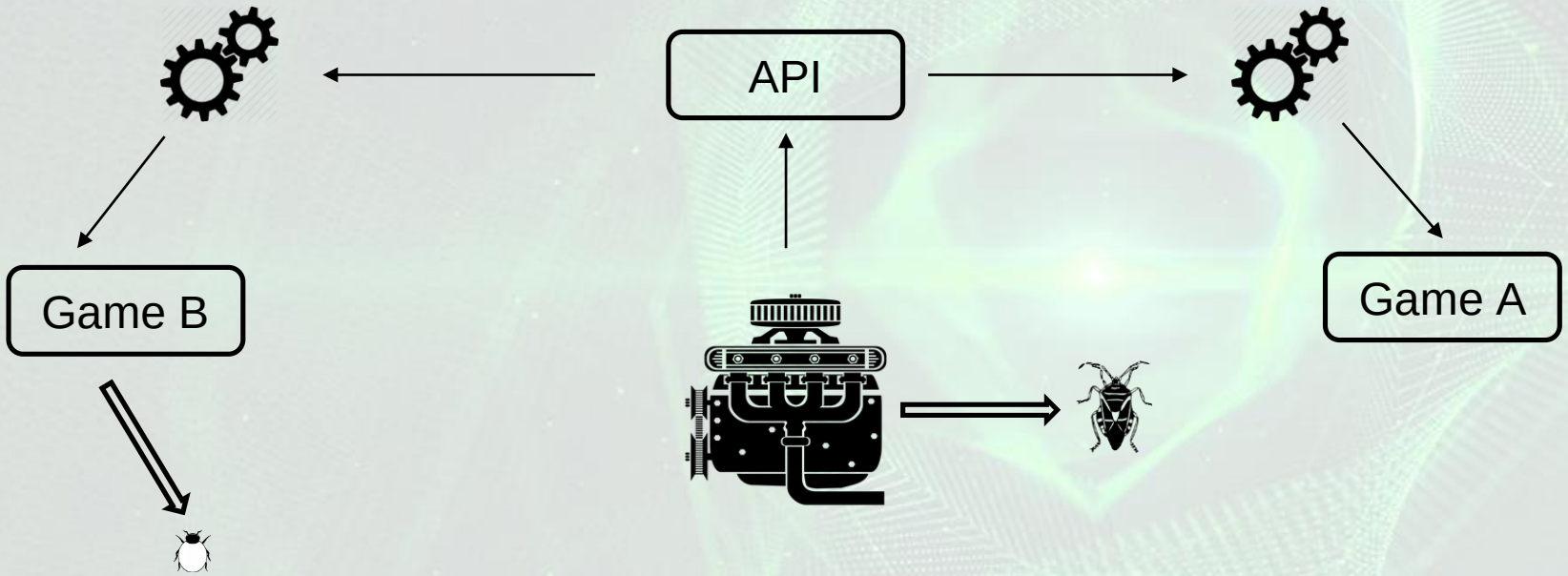
·虚幻引擎（UNREAL）：质量效应，战争机器…

·起源引擎（SOURCE）：CS， 军团要塞， DOTA2

·CRY ENGINE, UNITY3D…

漏洞？

- 由于性能等方面的原因，大部分的游戏引擎使用C/C++开发
- 而在游戏引擎中的漏洞，往往能影响同引擎开发的所有游戏

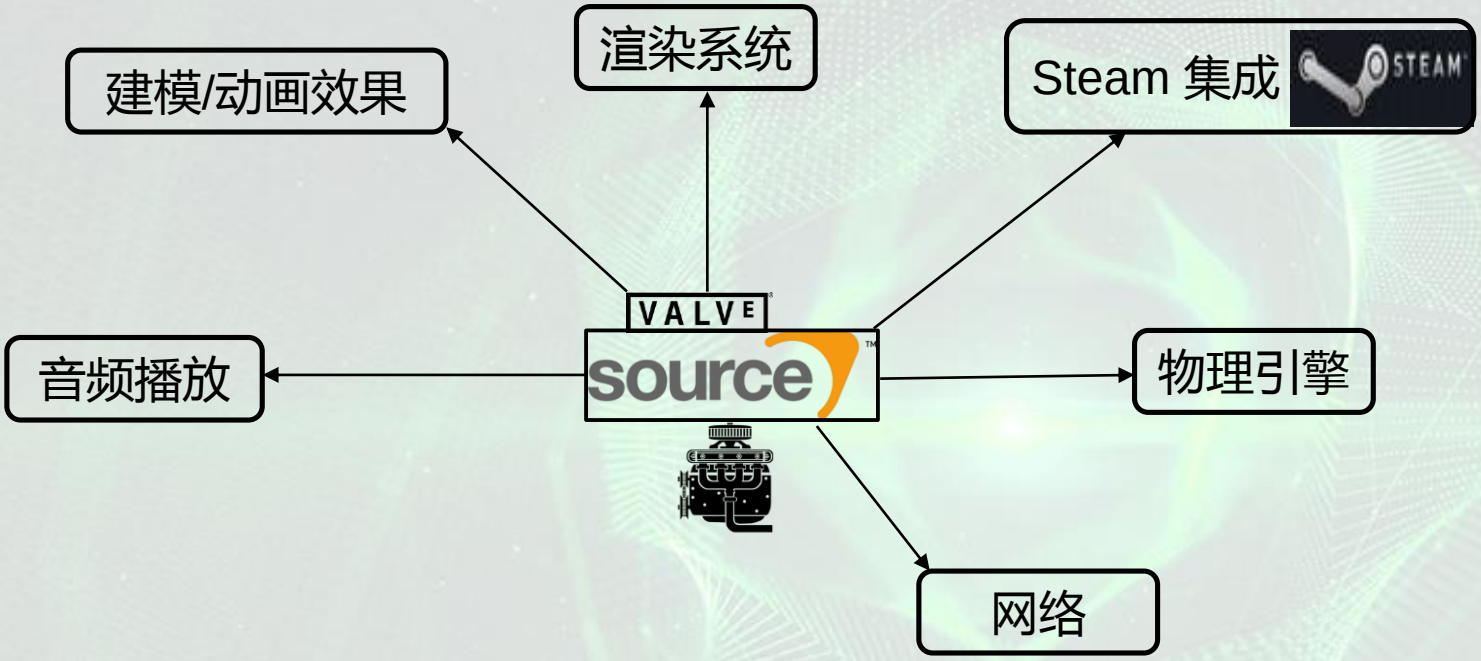


起源引擎

起源引擎

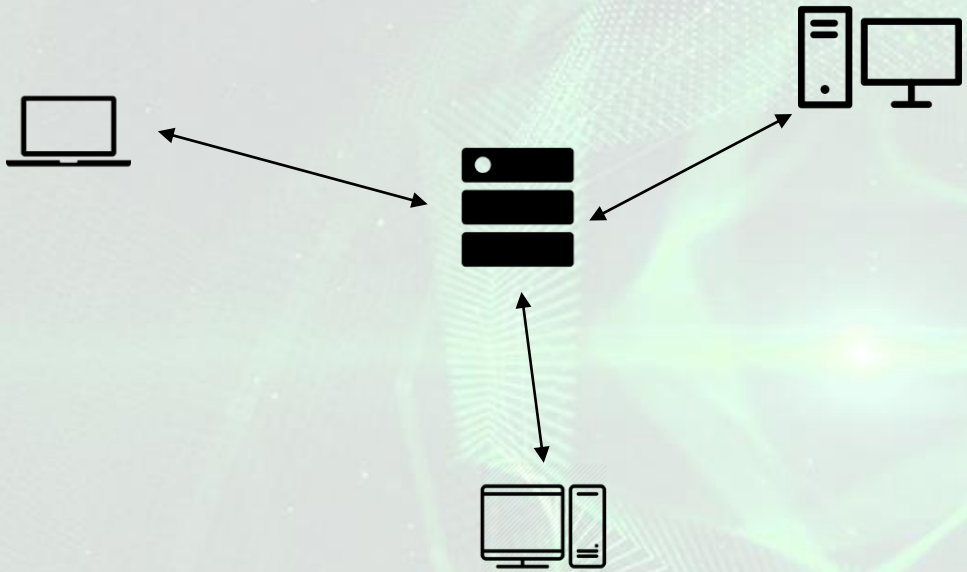
起源引擎最初为1998年为半条命开发的游戏引擎
作为开放授权引擎，使用范围很广

引擎构架与功能



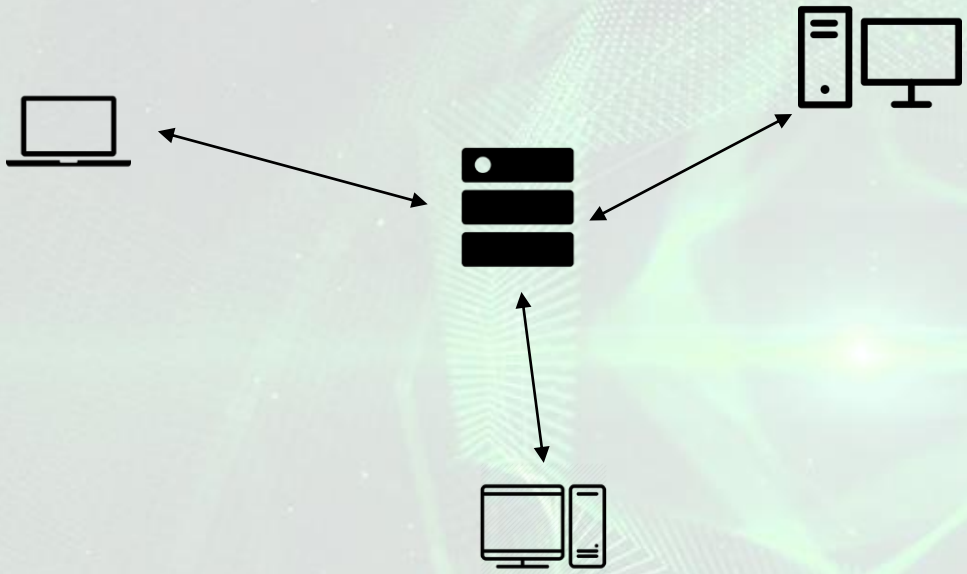
多人游戏网络构架

- 采用客户端-服务端网络模型
- 服务端负责游戏规则，玩家动作同步，建立世界等
- 客户端连接至服务端并且”遵守”规则
- 网络流量通过UDP/IP发送



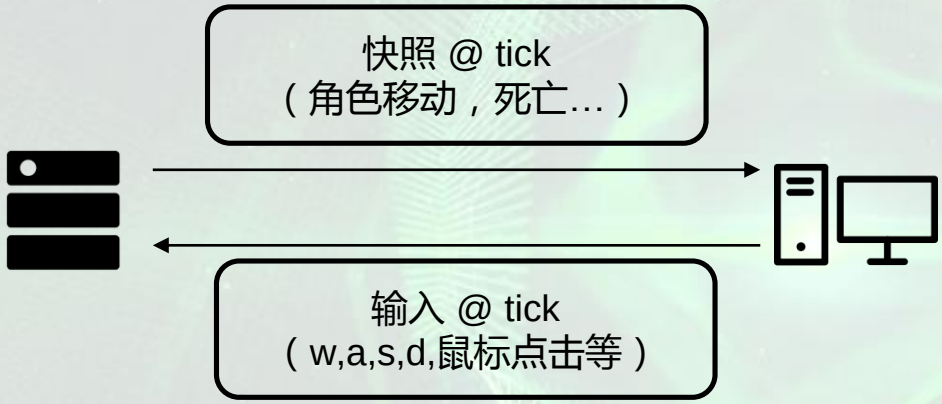
多人游戏网络构架

- 使用UDP以降低网络延时
- 起源引擎在UPD的基础上重新定义了“TCP”
- 压缩与解压缩
- 加密
- …各种底层的网络处理



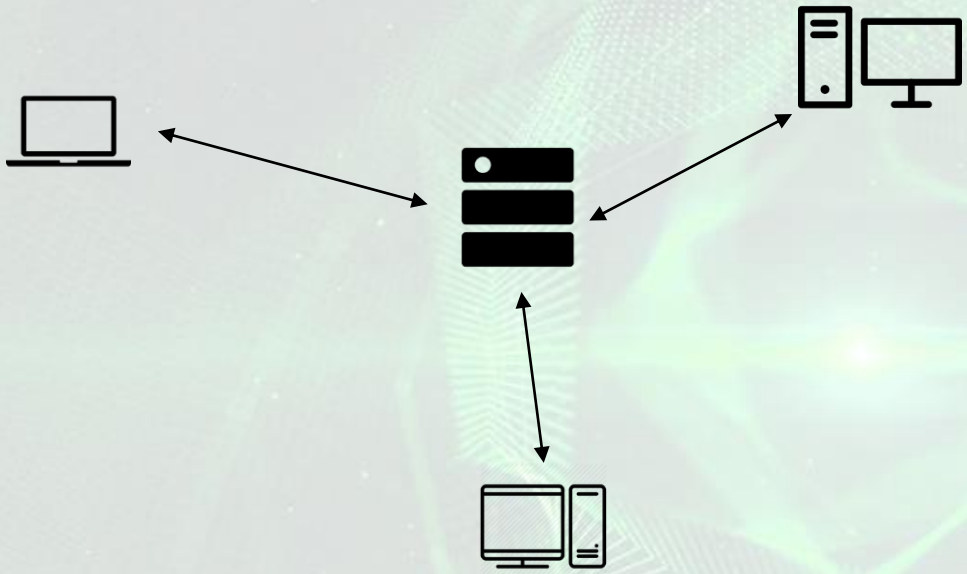
多人游戏网络构架

- 客户端与服务器同步数据的基本时间间隔：TICK
- 每一次TICK中，服务器会将游戏的快照发送给客户端
- 快照中包含着在此TICK中所有已经改变的实体
- 客户端只负责将用户的输入发给服务器



Message

- 起源引擎中，传输信息的基本单位
- 根据发送者分成三大类
- server message
- client message
- network message（客户端与服务端都会发送）



Network Message

客户端和服务端都可以发送的消息，以net_开头

- ➔ net_NOP
- ➔ net_Disconnect
- ➔ net_File
- ➔ net_LastControlMessage
- ➔ net_SplitScreenUser
- ➔ **net_Tick** ⓘ
- ➔ **net_StringCmd** ⓘ
- ➔ net_SetConVar
- ➔ net_SignonState

Client Message

由客户端发送，以clc_开头

- clc_ClientInfo
- clc_Move
- clc_VoiceData
- clc_BaselineAck
- clc_ListenEvents
- **clc_RespondCvarValue** ⓘ
- clc_FileCRCCheck
- clc_LoadingProgress
- clc_SplitPlayerConnect
- clc_ClientMessage
- clc_CmdKeyValues

Server Message

由服务端发送，以svc_开头

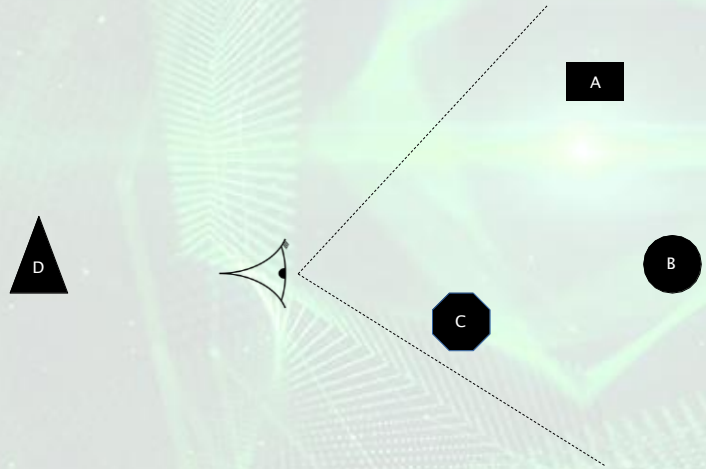
- svc_ServerInfo
- svc_SendTable
- **svc_CreateStringTable** ⓘ
- svc_UpdateStringTable
- **svc_UserMessage** ⓘ
- svc_EntityMessage
- svc_GameEvent
- **svc_PacketEntities** ⓘ
- svc_TempEntities
- svc_GameEventList
- **svc_GetCvarValue** ⓘ
- svc_CmdKeyValues
- ...

Server Message

- svc_UserMessage是游戏相关的一种特殊的message。由服务器发送，用来通知各种各样的游戏事件。
- 有着数量众多的子消息类型，每个游戏都有区别。
- 数量众多，但是漏洞通常没有办法其他游戏中存在
- E.g:
  - ➔ CS_UM_ShowMenu / TF_UM_ShowMenu
  - ➔ CS_UM_SayText
  - ➔ ...

Network Entities

- 实体（Entities）在客户端和服务端同时存在
- 起源引擎将几乎所有的东西都抽象为实体。服务器会保证所有客户端的实体保持同步
- 只有和客户端有关的实体才会被创建或者更新（可见的/可以听到的）
- 当客户端收到svc_PacketEntities消息时，会自动创建不存在的实体。
- 起源引擎最多可以同时持有2048个实体。
- 实体通常是在snapshots中发送的。



ConVars and ConCommands

- Console Variables 储存了服务端和客户端的各种配置参数，类型linux中的环境变量
- 有一些Console Variables会在客户端和服务端上同步
- Console Variables有各种属性，其中有一些可以被远端读取和设置
- e.g:
  - bot_dont_shoot: 设置后，bots将不会开火
  - host_map: 当前地图名字(只读，不可设置)

- Console Commands是一些命令用来实现各种附加功能
- 本质上是一些拥有特殊属性的ConVars
- e.g:
  - retry: 重新连接上一次连接的服务器
  - gods: 所有玩家无敌

ConVars and ConCommands

- svc_GetCvarValue： 服务端可以通过发送这个消息获取客户端的Cvar
- clc_RespondCvarValue ： 客户端通过这个消息给服务端返回Cvar
- net_StringCmd ： 客户端和服务端都可以发送这个消息让对方执行cmd

Message格式

- 起源引擎默认使用bitstreams来传输message消息
- 每个游戏可以自定义消息的传输格式
- 例如： CSGO使用google protocol代替bitstreams来传输消息
- 但是某些特殊的消息(例如svc_PacketEntities)依然使用bitsteams

起源引擎漏洞挖掘

起源引擎漏洞挖掘

攻击面有哪些？

- Message： 直接接受并处理网络数据，非常容易出问题
- engine.dll：引擎的核心，包括net_X 和svc_X 消息的处理函数
- client.dll：绝大部分游戏相关代码，所有svc_UserMessage子消息的处理函数

- Cvar：有数千个Cvar，许多可以由服务端来设置
- 地图解析：起源引擎的地图格式为BSP，客户端会自动的从服务端下载地图并加载
- 音频解析
- …

TF2中的整形溢出

- 服务端可以发送svc_CreateStringTable消息以创建一个字符串列表。
- 这个字符串列表记录了服务端经常发送的一些字符串
- 通过这个列表，服务端只需要发送一个index地址来代替字符串

TF2中的整形溢出

```
char CBaseClient::ProcessCreateStringTable(void *this, SVC_CreateStringTable *creatstrtab)
{
    ...
    int dataBits; // edx@2
    void *dest; // edi@8
    char *source; // esi@8
    ...
    int destLen; // [esp+30h] [ebp-Ch]@8
    int sourceLen; // [esp+38h] [ebp-4h]@6
    ...
    if ( creatstrtab->isCompressed )
    {
        dataIn = &creatstrtab->m_DataIn;
        ...
        destLen = READ32(dataIn);
        sourceLen = READ32(dataIn);
        ...
        dest = operator new[]((destLen + 3) & 0xFFFFFFFC);        // # 1
        source = operator new[]((sourceLen + 3) & 0xFFFFFFFC);    // # 2
        bf_read::ReadBits(dataIn, source, 8 * sourceLen);
        NET_BufferToBufferDecompress(dest, &destLen, source, sourceLen);
        ...
        operator delete[](dest);
        operator delete[](source);
    }
    ...
}
```

CS：GO中不存在此bug

CSGO中的数组越界写

- UM_ProcessSpottedEntityUpdate函数用以处理CS_UM_ProcessSpottedEntityUpdate消息
- 至于具体的作用，不用关心

起源引擎

CSGO中的数组越界写

```
int ProcessSpottedEntityUpdate(_BYTE *this, ProcessSpottedEntityUpdate_t *data)
{
    ...
    for ( i = 0; idx < data->numEntities; i = idx )
    {
        entitiesArray = data->entitiesArray;
        entName = 0;
        update = entitiesArray[idx];
        ent_idx = update->ent_idx;
        ...
        {
            ...
            {
                objidx = 0x1E0 * ent_idx;
                *&this_[objidx - 16] = 4 * update->origin_z;
                *&this_[objidx - 24] = 4 * update->origin_x;
                *&this_[objidx - 20] = 4 * update->origin_z;
                *&this_[objidx - 12] = 0;
                *&this_[objidx - 8] = update->angle_y;
                *&this_[objidx - 4] = 0;
            }
            ...
        }
        ...
    }
...
}
```

TF2中不存在此bug

起源引擎

CL_CopyNewEntity中的符号整形溢出

- 服务器会在每个tick中发送svc_PacketEntities消息以更新和创建那些已经发生改变的实体

```
    ...
    signed int newEntity; // edx@1
    ...
    IClientNetworkable *ent; // edi@3
    ...

    new
    if
    ent
    if
        return Host_Error("CL_CopyNewEntity: invalid class index (%d).\n", iClass);
    ...
    if ( ent )
    {
        v8 = ent->vtbl->someMethod(ent);
        ...
    }
    ...
}
```

```
IClientNetworkable * GetClientNetworkable(CClientEntityList *this, int index)
{
  return (&this->m_EntityCacheInfo)[2 * index];
}
```

起源引擎

CL_CopyNewEntity中的符号整形溢出

- 只要伪造一个object，越界读取回我们伪造的object，调用类虚方法的时候就可以控制eip
- 漏洞出现在起源引擎内核中，影响所有游戏

整形溢出利用

- 我们需要让entitylist.m_EntityCacheInfo[idx]指向我们伪造的结构体
- 让我们看看GetClientNetworkable 的汇编实现

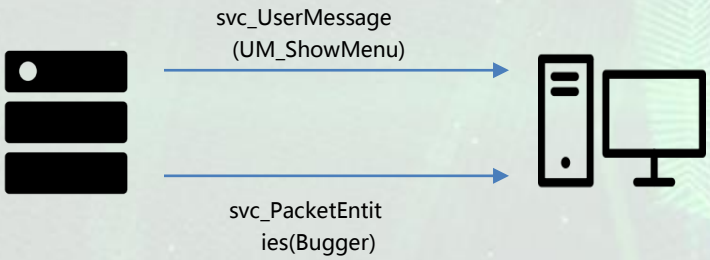
```
GetClientNetworkable proc near
index           = dword ptr  8
      push    ebp
      mov     ebp, esp
      mov     eax, [ebp+index]
      mov     eax, [ecx+eax*8+28h]
      pop     ebp
      retn    4
GetClientNetworkable endp
```

- 负数的index可以绕过长度检查。因为index会乘以8，所以可以构造任意index值
- 现在我们需要在.data 段构造数据

起源引擎

整形溢出利用

- 怎么在.data段存储数据？
- 让我们看看 UserMessages
- **UM_ShowMenu** 消息用来在客户端显示菜单
- 此消息会储存数据在client.dll 的数据段中
- 所以，现在的思路是发送**UM_ShowMenu** 消息来伪造C++类。然后触发 **svc_PacketEntities** 中的漏洞



client.dll
…
entitylist.m_EntityCacheInfo[]
…
fake_obj[] = {0x41414141,0x42424242, 0x43434343, ...}
…
…

整形溢出利用

- 由于aslr的关系，还需要考虑leak
- 让我们再看看CL_CopyNewEntity()函数

```
int __cdecl CL_CopyNewEntity(CEntityReadInfo *u, int iClass, int iSerialNum)
{
        ...
        ent = CL_CreateDLLEntity(u->m_nNewEntity, iClass, iSerialNum);
        if ( !ent )
        {
                ...
                return Host_Error("CL_ParsePacketEntities: Error creating entity" );
        }
        ...
        ent = (entitylist->vtbl->GetClientNetworkable)(u->m_nNewEntity);
        if ( !ent )
        {
                ...
                return Host_Error("CL_ParseDelta: invalid recv table for ent %d.\n", u->m_nNewEntity);
                ...
        }
}
```

起源引擎

整形溢出利用

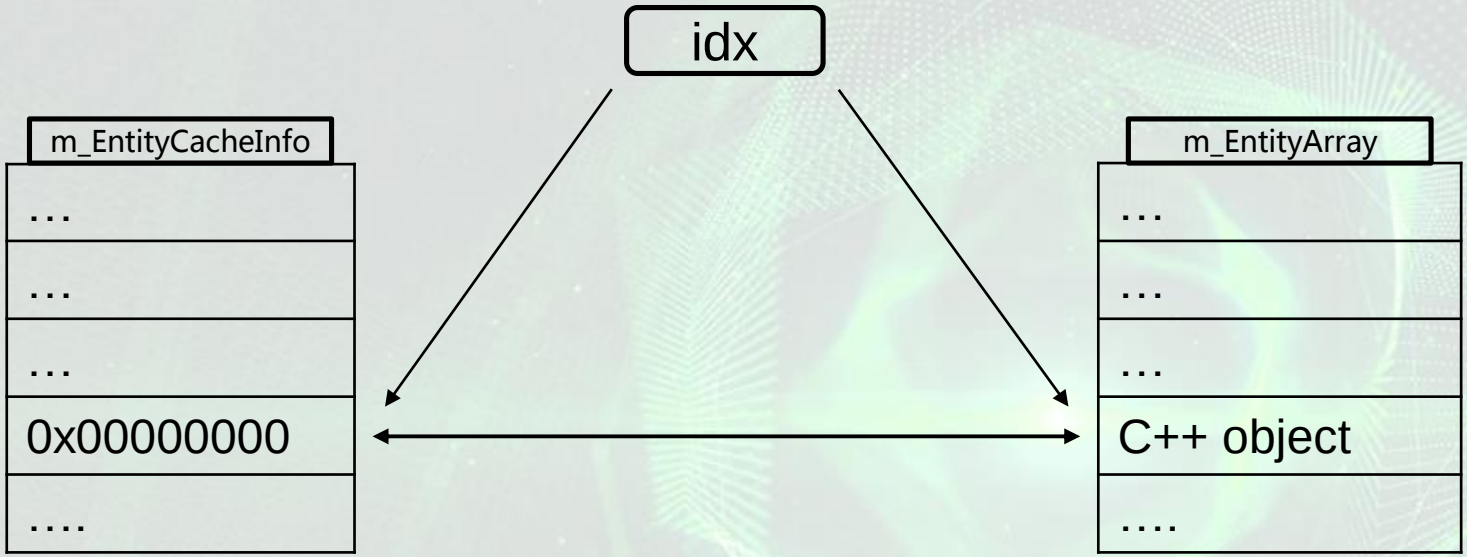
- 当实体不存在时（GetClientNetworkable 函数返回NULL），CreateDLLEntity 函数会自动创建一个新的实体
- 最后由AddEntityAtSlot函数添加实体
- 可以看到，最后实体被添加到的EntityArray数组中

```
int *CBaseEntityList::AddEntityAtSlot(unsigned int *EntityArray, IClientNetworkable *ent_object, int  index, int serial_num)
{
        unsigned int *object_ptr; // eax@1
        ...

        object_ptr = &EntityArray[4 * index + 1];
        *object_ptr =
        ent_object;
        if ( serial_num != -1 )
                EntityArray[4 * index + 2] = (unsigned int)serial_num;
        ...
}
```

整形溢出利用

- 有两个数组m_EntityArray 和m_EntityCacheInfo
- 有一个index
- 只要m_EntityCacheInfo中的值为NULL，就可以在m_EntityArray中创建一个C++对象

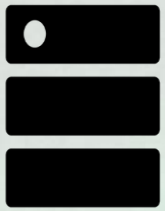


整形溢出利用

- 现在我们有了一个C++对象的指针
- 如果我们将它作为一个字符串读取回来，就可以leak出vtable从而绕过aslr
- 所以，怎么读？
- 答案是Cvars ☺
- 有许多的Cvar储存在clinet.dll 的.data段中，可以被覆盖
- Cvar有一个char * str_value 字段。只要将它替换为C++对象指针。然后用 **svc_GetCvarValue** 来读取

整形溢出利用

- 所以，现在的攻击思路是这样的



svc_PacketEntities (corrupt_cvar)

svc_GetCvarValue(corrupted_cvar)

svc_ResponCvarValue(corrupted_cvar)

svc_UserMessage(ShowMenuMessage)

svc_PacketEntities (pwn)

起源引擎

整形溢出利用

- 现在还有一个问题
- 由于整形溢出，之后的函数检查将会失败。
- Host_Error 函数将会断开客户端和服务端的连接
- 需要让客户端重新连接服务器

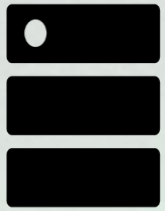
```
int __cdecl CL_CopyNewEntity(CEntityReadInfo *u, int iClass, int iSerialNum)
{
    ...
    ent = entitylist->vtbl->GetClientNetworkable(u->m_nNewEntity);
    if ( !ent )
    {
        ...
        return Host_Error("CL_ParseDelta: invalid recv table for ent %d.\n", u->m_nNewEntity);
        ...
    }
}
```

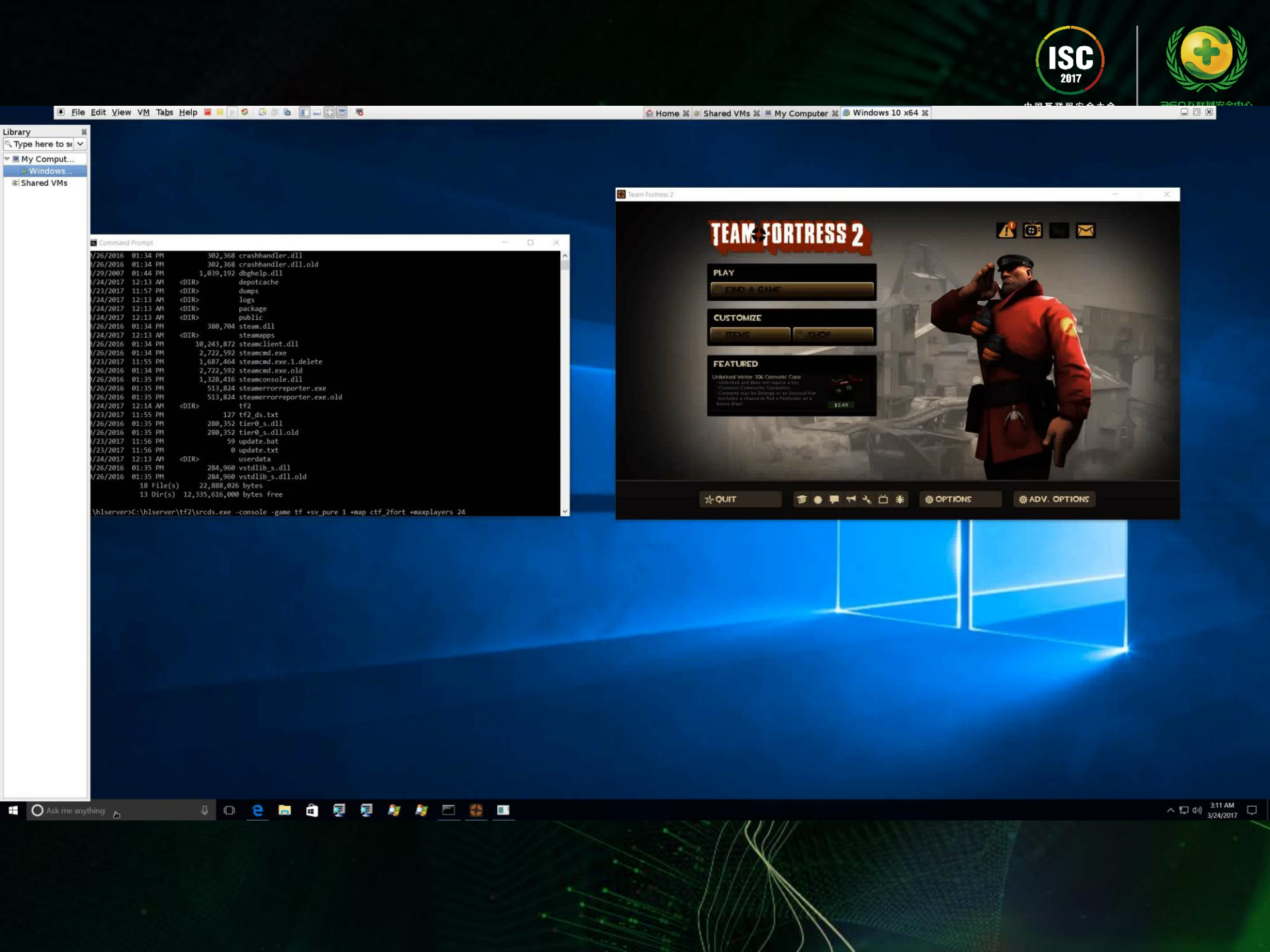
整形溢出利用

- 还记得ConCommands么？:)
- ConCommands中正好有这么一个命令
  - ➔ retry: 重新连接上一次连接的服务器
- 虽然没有办法给一个已经断开连接的客户端发送命令。
- 但是可以将command 和 svc_PacketEntities放在同一个数据包中发送
- 由于command会延时执行的原因，retry命令可以在服务器断开连接之后执行

整形溢出利用

- 所以，改进之后的攻击思路是这样的。




ISC
2017
File Edit View VM Tabs Help
Home
Shared VMs
My Computer
Windows 10 x64
Library
Type here to se
My Comput...
Windows...
Shared VMs
Command Prompt
9/26/2016 01:34 PM 302,368 crashhandler.dll
9/26/2016 01:34 PM 302,368 crashhandler.dll.old
9/29/2007 01:44 PM 1,039,192 dbghelp.dll
9/24/2017 12:13 AM <DIR> depotcache
9/23/2017 11:57 PM <DIR> dumps
9/24/2017 12:13 AM <DIR> logs
9/24/2017 12:13 AM <DIR> package
9/24/2017 12:13 AM <DIR> public
9/26/2016 01:34 PM 380,704 steam.dll
9/24/2017 12:13 AM <DIR> steamapps
9/26/2016 01:34 PM 10,243,872 steamclient.dll
9/26/2016 01:34 PM 2,722,592 steamcmd.exe
9/23/2017 11:55 PM 1,687,464 steamcmd.exe.1.delete
9/26/2016 01:34 PM 2,722,592 steamcmd.exe.old
9/26/2016 01:35 PM 1,328,416 steamconsole.dll
9/26/2016 01:35 PM 513,824 steamerrorreporter.exe
9/26/2016 01:35 PM 513,824 steamerrorreporter.exe.old
9/24/2017 12:14 AM <DIR> tf2
9/23/2017 11:55 PM 127 tf2_ds.txt
9/26/2016 01:35 PM 280,352 tier0_s.dll
9/26/2016 01:35 PM 280,352 tier0_s.dll.old
9/23/2017 11:56 PM 59 update.bat
9/23/2017 11:56 PM 0 update.txt
9/24/2017 12:13 AM <DIR> userdata
9/26/2016 01:35 PM 284,960 vstdlib_s.dll
9/26/2016 01:35 PM 284,960 vstdlib_s.dll.old
18 File(s) 22,888,026 bytes
13 Dir(s) 12,335,616,000 bytes free
\hlserver>C:\hlserver\tf2\srcds.exe -console -game tf +sv_pure 1 +map ctf_2fort +maxplayers 24
Team Fortress 2
TEAM FORTRESS 2
PLAY
FIND A GAME
CUSTOMIZE
ITEMS
SHOP
FEATURED
QUIT
OPTIONS
ADV. OPTIONS
Ask me anything
3:11 AM
3/24/2017

TODO

TODO

- 游戏引擎的安全性很低
- 不要连接陌生的游戏服务器
- Fuzzing 游戏引擎的某些模块，比如地图解析

谢　谢