ReVuln Ltd.
http://revuln.com
@revuln
info@revuln.com

Revision 3

STEAM VOIP SECURITY

BY LUIGI AURIEMMA

Overview and details about the security issues found in the Steam voice framework.

TABLE OF CONTENTS

Contents

Steam and Steamworks

STEAM

Steam[1] is the gaming platform developed by Valve Software and used by millions of players around the world to buy and play games, for multiplayer matchmaking and for its gaming-related social network.

STEAMWORKS API

The most interesting feature of Steam is its framework called Steamworks[2]. It contains the APIs used by most of the games available on this platform for integrating the following main features[3]:

- Stats & Achievements
- User Authentication & Ownership
- Multiplayer Matchmaking
- Community
- DLC and Content
- Peer-to-peer networking
- Cloud
- Anti-Cheat
- Voice chat
- DRM

The main aspect of the framework, from a security perspective, is it's role in increasing the attack surface of the games that use its API and making them remotely exploitable through its security vulnerabilities.

Additionally the framework and its operations are completely transparent to the final user (the player) who doesn't know what level of interaction is allowed from the other players or what are the external inputs that can be used to access and communicate with the framework.

The target of this security auditing has been just the Voice chat feature that allows the players to communicate via the voip system integrated in Steam and used on many games.

This research has been commissioned just by the developers of one of the games that use this Steamworks feature, *Epic Games*[4] for the *Unreal Engine 4* [5].

[1] http://steampowered.com
[2] http://www.steampowered.com/steamworks/
[3] https://partner.steamgames.com/documentation/api
[4] http://www.epicgames.com
[5] https://www.unrealengine.com

STEAM AND STEAMWORKS

VULNERABLE VERSIONS

This security auditing has been performed black-box (no Steam source code available) against the current versions of Steam and Steamworks available at the moment of the work:

Steam package versions: 1401381906

Steamclient.dll: 2.25.32.45

NON-VULNERABLE VERSONS

Steam package versions: 1404163764

Steamclient.dll: 2.30.30.94

Fixes available in Steam from 03 Jul 2014.

SECURITY MEASURES USED IN THE AFFECTED CODE

Quick considerations about the vulnerable Steamclient.dll:

- It supports DEP
- It doesn't support ASLR, which instead is supported by steamclient64.dll but is not used by Steam
- It's digitally signed
- The functions affected by stack-related vulnerabilities do not use stack cookies

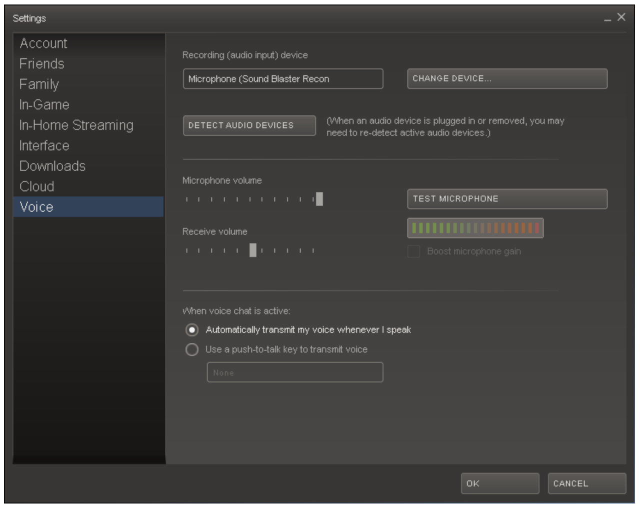
THE VOICE CHAT

The voice chat

The VOIP framework available in Steamworks can be used in two ways:

- As a voice chat with the Steam friends through the Steam chat interface
- In-game with the games that support it

Using the voice chat in Steam requires that the other endpoint is one of our friends and he accepts the voice chat[6].



Recording (audio input) device: Microphone (Sound Blaster Recon

(When an audio device is plugged in or removed, you may need to re-detect active audio devices.)

Microphone volume

Receive volume

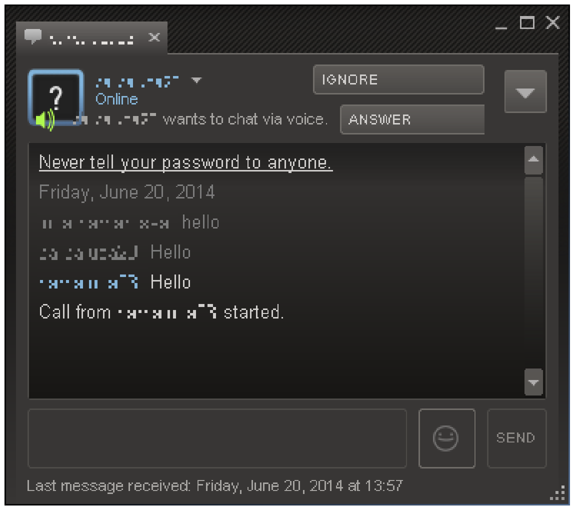
Boost microphone gain

When voice chat is active:

[✓ Automatically transmit my voice whenever I speak]

Use a push-to-talk key to transmit voice

None



[redacted]
Online
[redacted] wants to chat via voice.

Never tell your password to anyone.

Friday, June 20, 2014

[redacted] hello

[redacted] Hello

[redacted] Hello

Call from [redacted] started.

Last message received: Friday, June 20, 2014 at 13:57

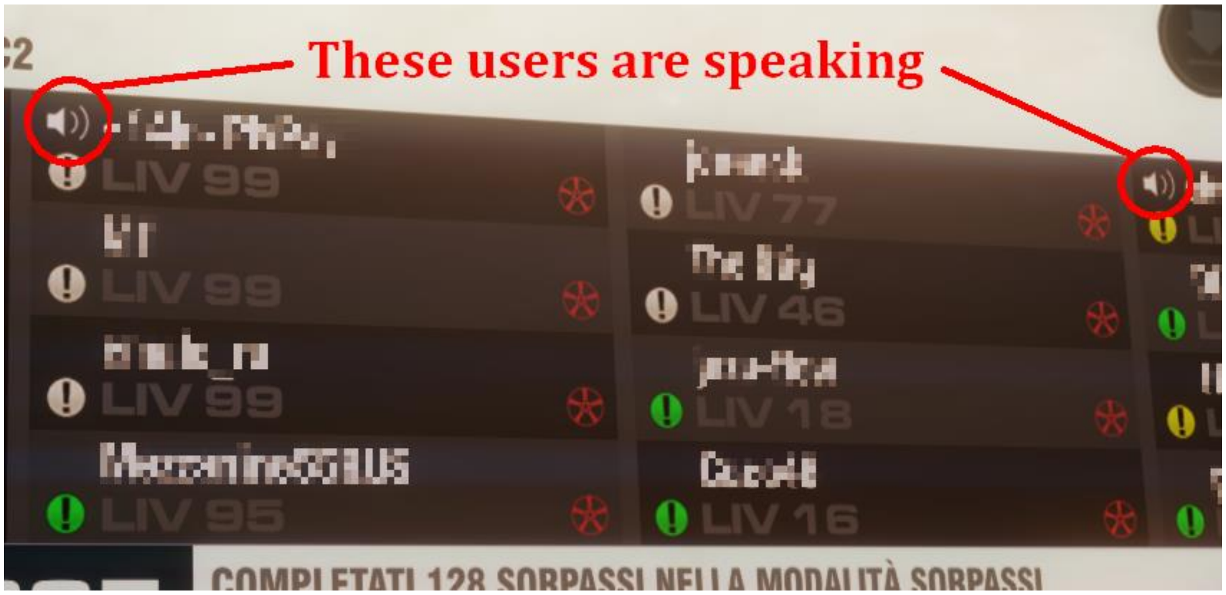
This sort of "restriction" doesn't involve the games where, instead, any player can speak to the other people inside the same lobby or server, often the members of the same team for those games in which there are two or more teams.

Additionally there is no need to configure settings, when a player joins a lobby both his voice and the one of the other players are already active.

[6] http://steamcommunity.com/sharedfiles/filedetails/?id=164655181

THE VOICE CHAT

The following is an example of users speaking while playing the game Grid 2[7].



The own voice can be captured via push-to-talk or automatically while talking, it depends by the default settings of the game and in both the cases it's necessary to have a certain level of input volume to activate the capturing, just like a squelch[8].

Receiving voice data is enabled by default on all the games and must be manually disabled or limited by the user if allowed by the game, for example in Grid 2 the game gives automatically "voice to all" everytime we join and change a lobby.

That means any user playing on the same server, lobby or team of the attacker will be targeted by any malicious voip data broadcasted by the attacker through the server or peer-to-peer.

[7] http://store.steampowered.com/app/44350/

[8] http://en.wikipedia.org/wiki/Squelch

Technical overview of the voice chat

STEAMCLIENT.DLL AND THE API

Technically the in-game voice chat works in the following way:

- All the code necessary to handle the data is located in steamclient.dll and steamclient64.dll
- This DLL is located in the Steam folder and a copy, got from the Steamworks framework used by the game, is located in the game folder
- The game loads the DLL of the Steam folder
- In this case the DLL works as a wrapper interfaced via IPC (events, shared memory and named pipes) to the running Steam.exe process
- So when the game calls an API, all the operation is executed inside the Steam.exe process through the steamclient.dll loaded in it (both Steam and the game use this DLL)
- A security vulnerability in the API compromises Steam and causes the freezing of the game which is waiting a reply from the IPC interface
- Steam offers some APIs to use its network code but it's up to the game to use it or their own protocol to transmit and receive the audio data

The IsteamUser APIs that handle the voice data are very easy to use:

- GetVoice for capturing and compressing the voice data from the microphone
- DecompressVoice for decoding the received audio data in 16-bit, signed integer PCM format

What a game does is using its own protocol for sending and receiving the audio data without touching or modifying the content of the data generated by GetVoice or received from the network. The role of a server is just broadcasting the received data "as is" to the other clients.

The clients call DecompressVoice on the received data to decode it.

There are no limitations about the size of the voip data, so it's possible to use DecompressVoice with chunks of any size and it's all up to the game. For example Unreal Engine 4 uses a buffer of 8192 bytes while Portal 2 and Counter Strike Global Offensive use 18432, Team Fortress 2 and Half-Life 2 use 2048 and all the others like SteamworksExample project and Grid 2 use 1024.

So, just to recap, if an attacker sends malformed voip data to the chat of a game, it's Steam that will be exploited.

Please note that the Steam chat doesn't use GetVoice API to capture the audio, but it uses DecompressVoice for decoding the incoming data and so it's vulnerable to the vulnerabilities described in this paper.

TECHNICAL OVERVIEW OF THE VOICE CHAT

DECOMPRESSVOICE

The DecompressVoice API is our target because the data sent by the other players is received and passed to this function “as is”.

The following are the prototype and the comments from Steamworks:

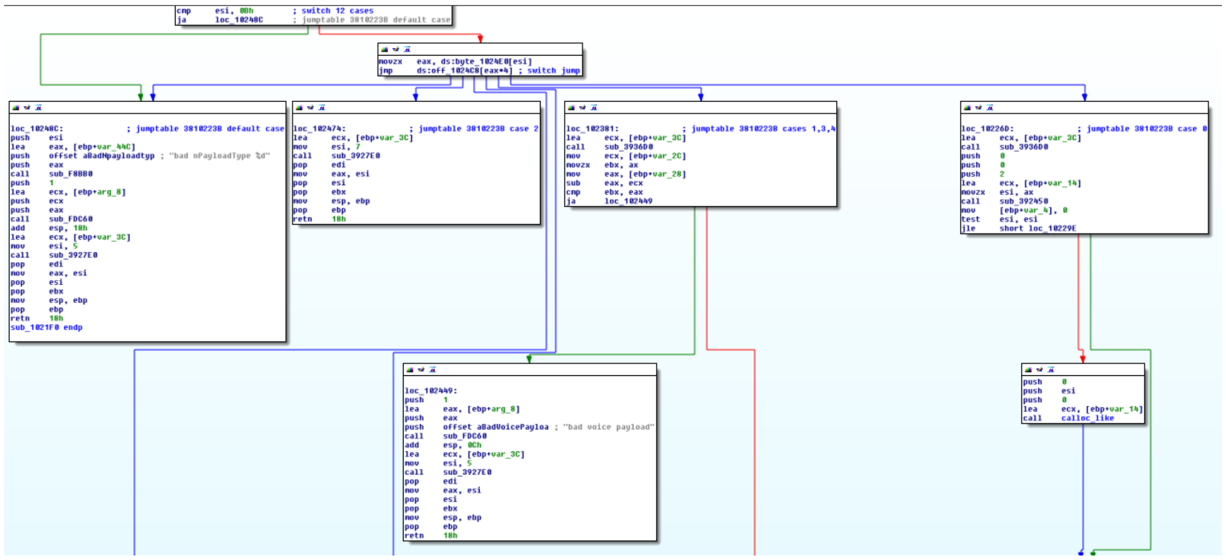
```
// Decompresses a chunk of compressed data produced by GetVoice().
// nBytesWritten is set to the number of bytes written to pDestBuffer unless the return value is
   k_EvoiceResultBufferTooSmall.
// In that case, nBytesWritten is set to the size of the buffer required to decompress thegiven
// data. The suggested buffer size for the destination buffer is 22 kilobytes.
// The output format of the data is 16-bit signed at the requested samples per second.
// If you’re upgrading from an older Steamworks API, you’ll want to pass in 11025 to nDesiredSampleRate

virtual EvoiceResult DecompressVoice(

        const void *pCompressed,            < INPUT DATA

        uint32 cbCompressed,                < SIZE OF INPUT DATA

        void *pDestBuffer,                  > OUTPUT BUFFER

        uint32 cbDestBufferSize,            > MAXIMUM SIZE OF OUTPUT BUFFER

        uint32 *nBytesWritten,              > DECODED BYTES WRITTEN

        uint32 nDesiredSampleRate           | STEAMWORKS LIMITS IT TO MAX 49000

        ) = 0;
```

Our input data is not just raw compressed audio data, it contains a header and opcodes with various fields.

The main header is composed by a 32bit field followed by three flags packed as bitfields:

```
        typedef struct {

                uint32 id;

                uint32 flag1:20;

                uint32 flag2:4;

                uint32 flag3:8;

        }
```

This header is followed by a sequence of opcodes called nPayloadType:



```
cmp     esi, 0Bh            ; switch 12 cases
ja      loc_10248C          ; jumptable 3810223B default case
```

```
movzx   eax, ds:byte_1024E0[esi]
jmp     ds:off_1024C8[eax*4] ; switch jump
```

```
loc_10248C:                 ; jumptable 3810223B default case
push    esi
lea     eax, [ebp+var_44C]
push    offset aBadNpayloadtyp ; "bad nPayloadType %d"
push    eax
call    sub_F88B0
push    1
lea     ecx, [ebp+arg_8]
push    ecx
push    eax
call    sub_FDC60
add     esp, 18h
lea     ecx, [ebp+var_3C]
mov     esi, 5
call    sub_3927E0
pop     edi
mov     eax, esi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn    18h
sub_1021F0 endp
```

```
loc_102474:                 ; jumptable 3810223B case 2
lea     ecx, [ebp+var_3C]
mov     esi, 7
call    sub_3927E0
pop     edi
mov     eax, esi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn    18h
```

```
loc_102381:                 ; jumptable 3810223B cases 1,3,4
lea     ecx, [ebp+var_3C]
call    sub_3936D0
mov     ecx, [ebp+var_2C]
movzx   ebx, ax
mov     eax, [ebp+var_28]
sub     eax, ecx
cmp     ebx, eax
ja      loc_102449
```

```
loc_10226D:                 ; jumptable 3810223B case 0
lea     ecx, [ebp+var_3C]
call    sub_3936D0
push    0
push    0
push    2
lea     ecx, [ebp+var_14]
movzx   esi, ax
call    sub_392450
mov     [ebp+var_4], 0
test    esi, esi
jle     short loc_10229E
```

```
loc_102449:
push    1
lea     eax, [ebp+arg_8]
push    eax
push    offset aBadVoicePayloa ; "bad voice payload"
call    sub_FDC60
add     esp, 0Ch
lea     ecx, [ebp+var_3C]
mov     esi, 5
call    sub_3927E0
pop     edi
mov     eax, esi
pop     esi
pop     ebx
mov     esp, ebp
pop     ebp
retn    18h
```

```
push    0
push    esi
push    0
lea     ecx, [ebp+var_14]
call    calloc_like
```

At the end of the data is located the 32bit CRC, a classical checksum calculated on the whole data before it.

NPAYLOADTYPE 0

Arguments: uint16 samples

Used as a way to create silence by transmitting only the 16bit number containing the desired amount of samples to fill.

It uses the same function of nPayloadType 1, 3 and 4 by passing it an allocated buffer filled with zeroes used as input data and choosing the codec 3 (raw PCM).

The temporary buffer used by this function to contain the decompressed chunk is located on the stack and has a size of 16428 bytes.

NPAYLOADTYPE 1, 3 AND 4

Arguments: uint16 samples followed by data

Type 1 is a codec no longer available, probably Miles[9].

Type 3 is the uncompressed 16bit PCM data, copied "as is" to the destination buffer.

[9] http://www.radgametools.com/miles.htm

TECHNICAL OVERVIEW OF THE VOICE CHAT

Type 4 is the Silk[10] codec, the function performs its initialization using the default samplerate (11025) or the one specified by nPayloadType 11.

The Silk codec has been introduced in Steam since 2011[11] replacing Miles. Steam uses the SILK SDK provided by Skype.

These opcodes check the number of samples specified in the packet to avoid that what is specified is more than the data available in the chunk.

NPAYLOADTYPE 2

Arguments: none

End of voip data.

NPAYLOADTYPE 10

Arguments: uint8 bytes[2]

Unused.

NPAYLOADTYPE 11

Arguments: uint16 samplerate

Used for specifying the sample rate (the frequency in hertz) of the input data.

[10] http://en.wikipedia.org/wiki/SILK

[11] http://store.steampowered.com/news/5100/

Security issues

STACK CONTROLLED CORRUPTION

By using nPayloadType 0 we can decide the 16bit size of an input buffer containing zeroes that will be copied directly in a stack buffer of 16428 bytes:

```
.text:00102291                 push    0
.text:00102293                 push    esi
.text:00102294                 push    0
.text:00102296                 lea     ecx, [ebp+var_14]
.text:00102299                 call    calloc_like
.text:0010229E                 lea     eax, [esi+esi]
.text:001022A1                 push    eax             ; size_t
.text:001022A2                 push    0               ; int
.text:001022A4                 push    [ebp+var_10]    ; void *
.text:001022A7                 call    _memset
...
; int __stdcall sub_101F30(void *, size_t, int, int)
.text:00101F30                 push    ebp
.text:00101F31                 mov     ebp, esp
.text:00101F33                 mov     eax, 402Ch      ; 16428
.text:00101F38                 call    __alloca_probe
...
.text:00101F8E                 mov     eax, [ebp+arg_8]
.text:00101F91                 push    edi
.text:00101F92                 mov     edi, ebx
.text:00101F94                 shr     edi, 1
.text:00101F96                 cmp     eax, 3
.text:00101F99                 jnz     short loc_101FEA
.text:00101F9B                 push    ebx             ; size_t
.text:00101F9C                 push    [ebp+arg_0]     ; void *
.text:00101F9F                 lea     eax, [ebp+var_402C]
.text:00101FA5                 push    eax             ; void *
.text:00101FA6                 call    _memmove_0
```

The problem is caused by the function that handles the data of the codec because it doesn't check if the input data is bigger than the available stack buffer, resulting in a stack-based buffer-overflow.

The possibility of specifying the exact number of zeroes to write on the stack and the lack of stack cookies, allow an attacker to modify the lower part of the saved addresses, for example by using "(16428 – 0x4) / 2" as number of samples.

Anyway code execution doesn't seem possible on the Windows version we tested.

STACK-BASED BUFFER-OVERFLOW (GAME DEPENDENT)

There is also a way to exploit the previous vulnerability using controlled content.

The only check performed by nPayloadType 1, 3 and 4 is related to the amount of samples and the size of the input data, but there are no checks performed by the function seen before.

So if a game can receive an audio chunk of more than 16428 bytes, it's possible to exploit the relative stack-based buffer-overflow using the provided data. The Steamworks documentation recommends to use a buffer of 8 kilobytes or **larger** for the compressed audio collected with GetVoice.

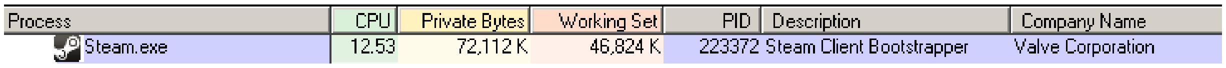
Portal 2 and Counter Strike Global Offensive are some of the games tested by us that support packets bigger than that stack buffer size, exactly 18432 bytes. Other games may be vulnerable too.

SAMPLERATE ENDLESS LOOP

With nPayloadType 11 we can set the desired sample rate of our audio data and it can be any number between 0 and 65535. If we set the sample rate to zero we are able to cause an endless loop in the following cycle:

```
.text:001024F0 ; int __stdcall sub_1024F0(int, int, double)
...
.text:00102523 loc_102523:                             ; CODE XREF: sub_1024F0+A8
.text:00102523                 fld     st
.text:00102525                 call    __ftol2_sse
.text:0010252A                 mov     esi, eax
.text:0010252C                 sub     esp, 8
.text:0010252F                 movsx   ecx, word ptr [ebx+esi*2]
.text:00102533                 mov     [ebp+arg_4], ecx
.text:00102536                 fild    [ebp+arg_4]
.text:00102539                 fstp    [ebp+var_18]
.text:0010253C                 fstp    [esp+34h+var_34] ; double
.text:0010253F                 call    _floor
.text:00102544                 fsubr   [ebp+var_8]
.text:00102547                 movsx   eax, word ptr [ebx+esi*2+2]
.text:0010254C                 add     esp, 8
.text:0010254F                 mov     [ebp+arg_4], eax
.text:00102552                 fild    [ebp+arg_4]
.text:00102555                 fstp    [ebp+var_10]
.text:00102558                 fld     [ebp+var_10]
.text:0010255B                 fld     [ebp+var_18]
.text:0010255E                 fsub    st(1), st
.text:00102560                 fxch    st(2)
.text:00102562                 fmulp   st(1), st
.text:00102564                 faddp   st(1), st
.text:00102566                 call    __ftol2_sse
.text:0010256B                 movzx   eax, ax
.text:0010256E                 lea     ecx, [edi+30h]
.text:00102571                 mov     [ebp+arg_0], eax
.text:00102574                 lea     eax, [ebp+arg_0]
.text:00102577                 push    2               ; int
.text:00102579                 push    eax             ; void *
.text:0010257A                 call    sub_399FD0
.text:0010257F                 fld     [ebp+var_8]
.text:00102582                 fadd    [ebp+arg_8]
.text:00102585                 add     dword ptr [edi+7D5Ch], 2
.text:0010258C                 fst     [ebp+var_8]
.text:0010258F                 fld     [ebp+var_20]
.text:00102592                 fxch    st(1)
.text:00102594                 fcomi   st, st(1)
.text:00102596                 fstp    st(1)
.text:00102598                 jb      short loc_102523
```

The Steam process remains freezed with the assigned core of the CPU at 100% and must be killed from the Task Manager:



Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
Steam.exe	12.53	72,112 K	46,824 K	223372	Steam Client Bootstrapper	Valve Corporation

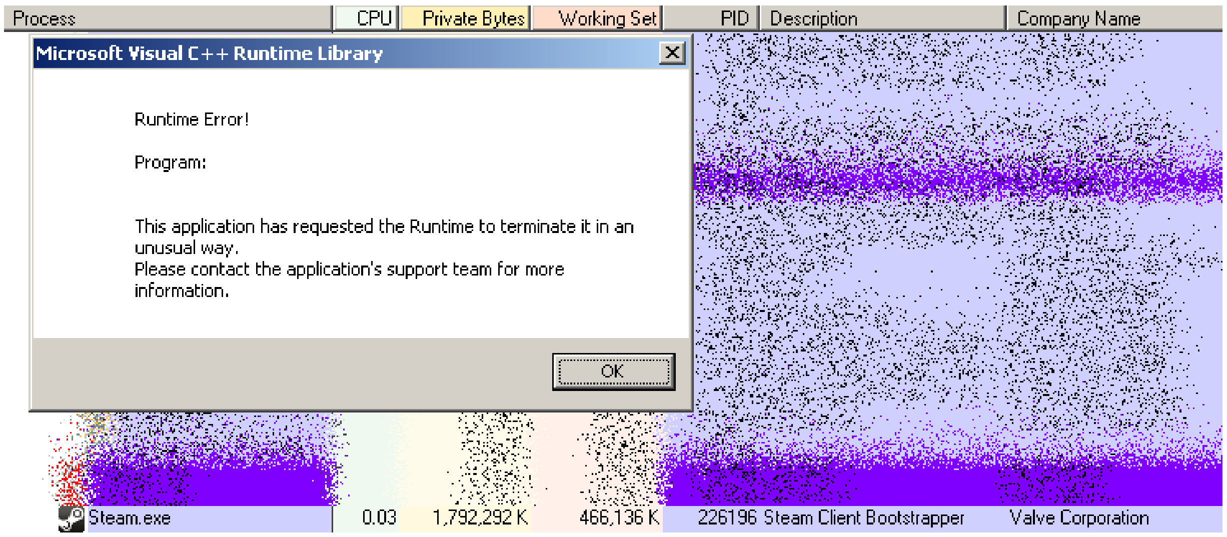
SECURITY ISSUES

MEMORY CONSUMPTION

The first 32bit field of the voip header is used as an ID, and the Steam process allocates new resources everytime a new ID is parsed.

These resources remain allocated for all the time Steam is running and an attacker can saturate all the memory of the Steam process that is limited to less than 2 Gigabytes since it's a 32bit program.

When there is no longer memory available for the process, Steam terminates with the following error message:



Runtime Error!

Program:

This application has requested the Runtime to terminate it in an unusual way.
Please contact the application's support team for more information.

Steam.exe 0.03 1,792,292 K 466,136 K 226196 Steam Client Bootstrapper Valve Corporation

SECURITY IMPACT

Security Impact

Considering the types of security issues found during this auditing, we think that no previous security assessment has been performed on such code.

The minimum risk derived from these issues is a Denial of Service affecting not only the Steam process but also the target game and any other game using the Steamworks API because all the Steamworks operation are handled by the Steam process.

Particularly interesting is the endless loop.

Code execution may be possible.

The most critical part of these issues is that it's not needed to develop a proof-of-concept or an exploit specific for the target game.

In fact we created a very simple proof-of-concept consisting of a DLL that is injected in the running Steam.exe process and replaces the original GetVoice function (the one called via IPC) with ours that fills the buffer with the desired malformed data.

The result is that any game supporting the Steam voip can automatically exploit any remote player reachable by the malformed voip data.

The proof-of-concept for steamclient.dll 2.25.32.45 is available as source code and pre-compiled dll:

- http://revuln.com/files/steamute_src.zip
- http://aluigi.org/poc/steamute.zip

Read the header of *steamute.c* for information and details on how to use it.

FAQ

How much critical are these issues?

Denial of Service and possible code execution from remote without user interaction.

The issues affect the Steam process and, as side effect, cause a Denial of Service in the game.

A vulnerability exploitable inside the Steam process is much worst than one affecting the game due to the single target (Steam.exe) for tuning the own code and the amount of personal information and interaction possible through this platform in case of possible code execution.

Do I need to authorize a player to be vulnerable?

No, all the games automatically allow other players to send voice data, in some games the attacker must be in the same team of the victim.

Blacklisting or whitelisting a player is optional and may not be implemented in some games.

Do I need to have Steam running to be vulnerable?

Steam is launched automatically by the supported games.

Is it easy for an attacker to exploit these issue?

Our proof-of-concept consists of some lines of code injected in the Steam process and being able to test automatically any game that uses DecompressVoice.

So, yes, it's very easy.

Additionally doesn't matter if the game is compiled as 64bit because Steam is 32bit and so uses only steamclient.dll, not steamclient64.dll.

Why Steam crashes when I exploit these vulnerabilities?

That's caused by how Steam and Steamworks operate: the APIs of Steamworks communicate with the Steam process via IPC so if you exploit these vulnerabilities when you are playing a game like Grid 2 the effect will be the crash of Steam.exe and the game no longer responding.

FAQ

Is the game *** vulnerable?

All the games that rely on Steamworks for handling the voice data are vulnerable, and obviously also Steam itself through its integrated voice chat.

Currently we don't have an exaustive list of games using such feature but some of the most played games[12] we tested that use the DecompressVoice API are the following:

- Half-Life 2 *(basically any Valve game, with the only exception of DOTA 2)*
- Portal 2
- Counter Strike: Global Offensive / Counter Strike / Counter Strike: Source / Garry's Mod
- Left for Dead 2
- Team Fortress 2
- Borderlands 2
- Grid 2
- Dirt 3 Showdown
- the recent games of the Worms series
- Unreal Engine 4

Is DOTA 2 vulnerable?

No, DOTA 2 doesn't use the DecompressVoice API of Steamworks.

I'm a game developer who uses Steamworks and my software uses many secure memory protections.

The role of the game is just passing the malformed audio data received from the network to the Steam.exe process via IPC so any memory protection inside the game is completely useless.

My game uses version *** of Steamworks, am I vulnerable?

The version of Steamworks used by the game doesn't matter, the vulnerability is exploited in the steamclient.dll library loaded by the Steam.exe process.

Please check the "Vulnerable versions" section of this paper to know what's the latest known version of steamclient.dll that is affected by these issues. The current version of Steam is fixed.

[12] http://store.steampowered.com/stats/?snr=1_steam_4__110

CHANGELOG

Changelog

20 Jun 2014	vulnerabilities reported to Epic Games and then Valve
25 Jun 2014	vulnerabilities fully fixed in Steam beta client
03 Jul 2014	fixes implemented in the stable Steam client
04 Jul 2014	public release of this document

COMPANY INFORMATION

Company Information

ReVuln Ltd.
Level 3, Theuma House, 302, St.Paul Street,
Valletta VLT1213
Malta
http://revuln.com
@revuln
info@revuln.com