Hello, Good afternoon guys! Thank you for being here.

I’m Jeonghoon Shin from South Korea.

Today, I’m talking about the Journey(져얼-니) of browser hacking with ANGLE.

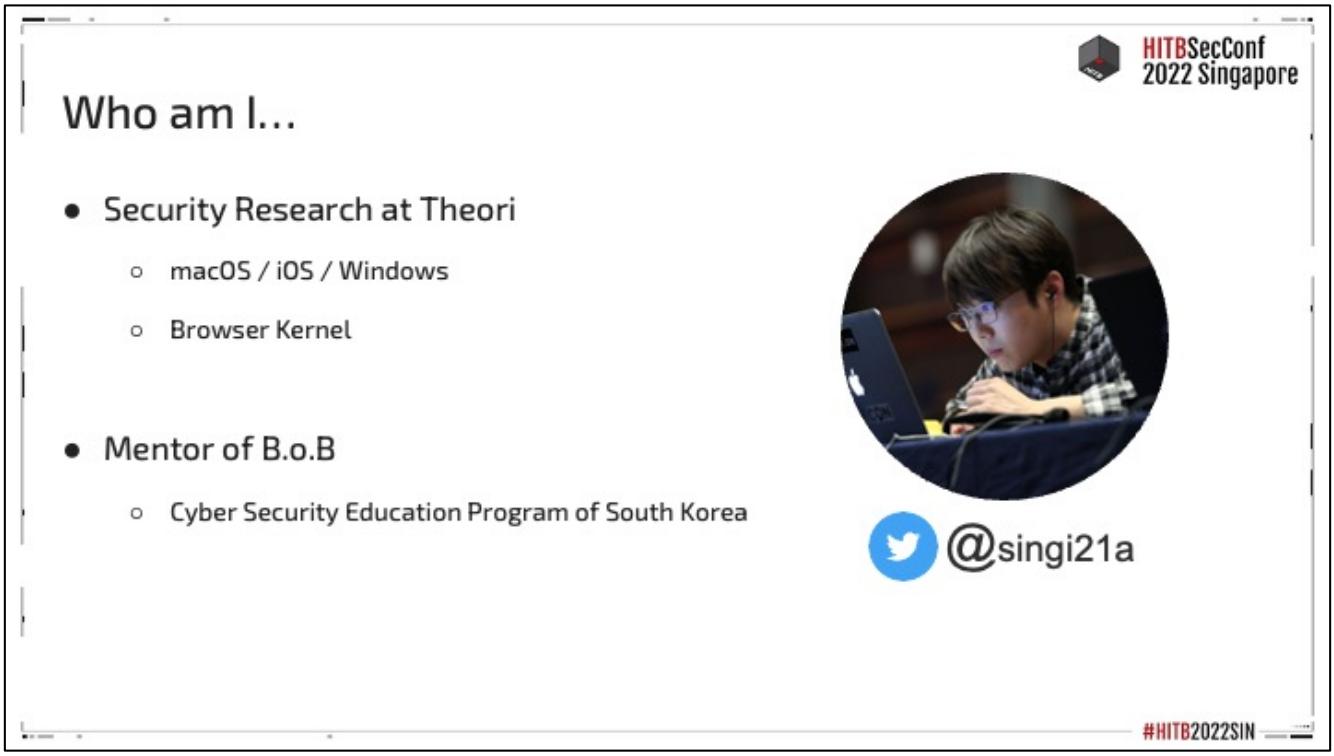
But before we start, Let me tell you one of my concerns. (렛 미 텔유 원오브 마이 컨설스) (….or I have one concern.)

That is … I can't speak English well.

If I can't make eye contact with you guys, because I'm reading my script.

It's not because I don't like you. Please be kind to me.

So, let me introduce myself.

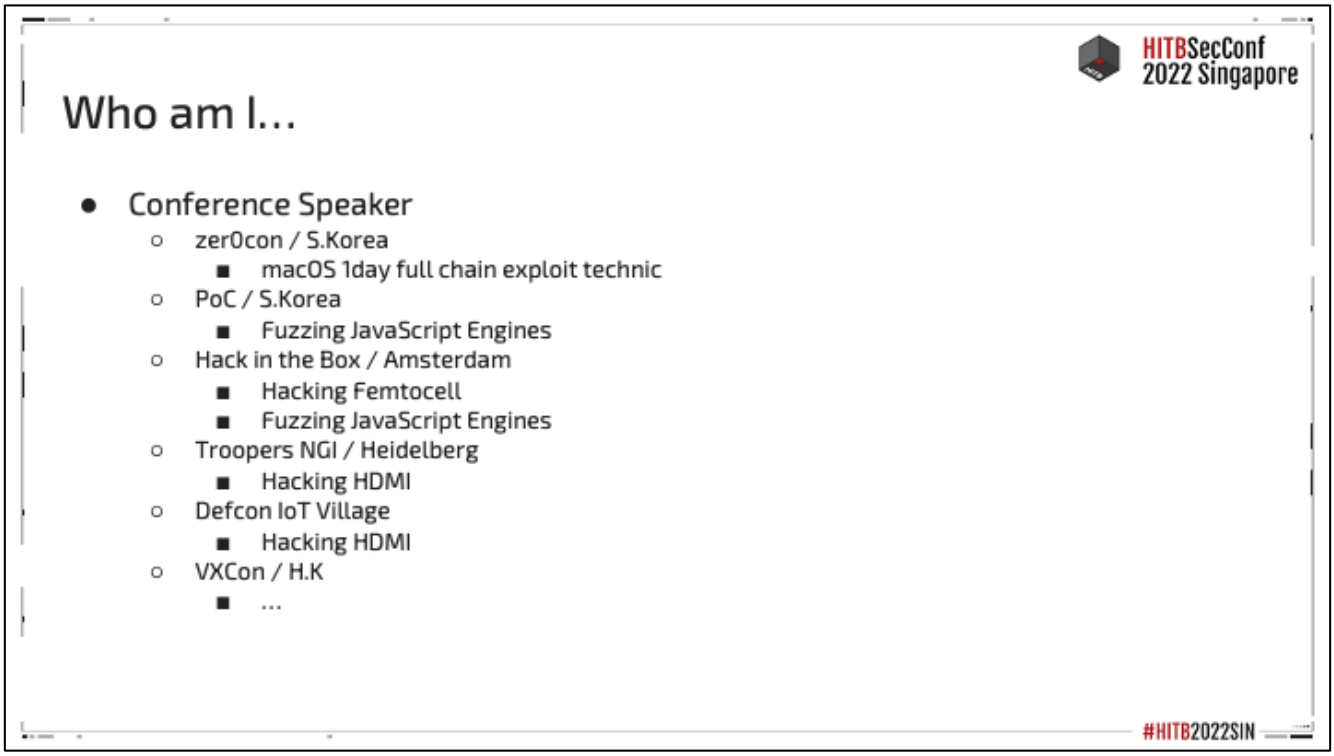


I'm working on Theori for 5 years, Also my Position is Security Researcher, and I Usually focusing on Browser and Kernel Research.

Also, Im mentor of the Best of the Best Program, called B.o.B.

B.o.B is Cyber Security Education Program in / South Korea, It has been operating for 11 years, and has Educating about 1,400(폴틴헌드레드) students.

In CTF competitions(컴패티-션스) like Defcon / You can see the many teams that / B.o.B students belong to.

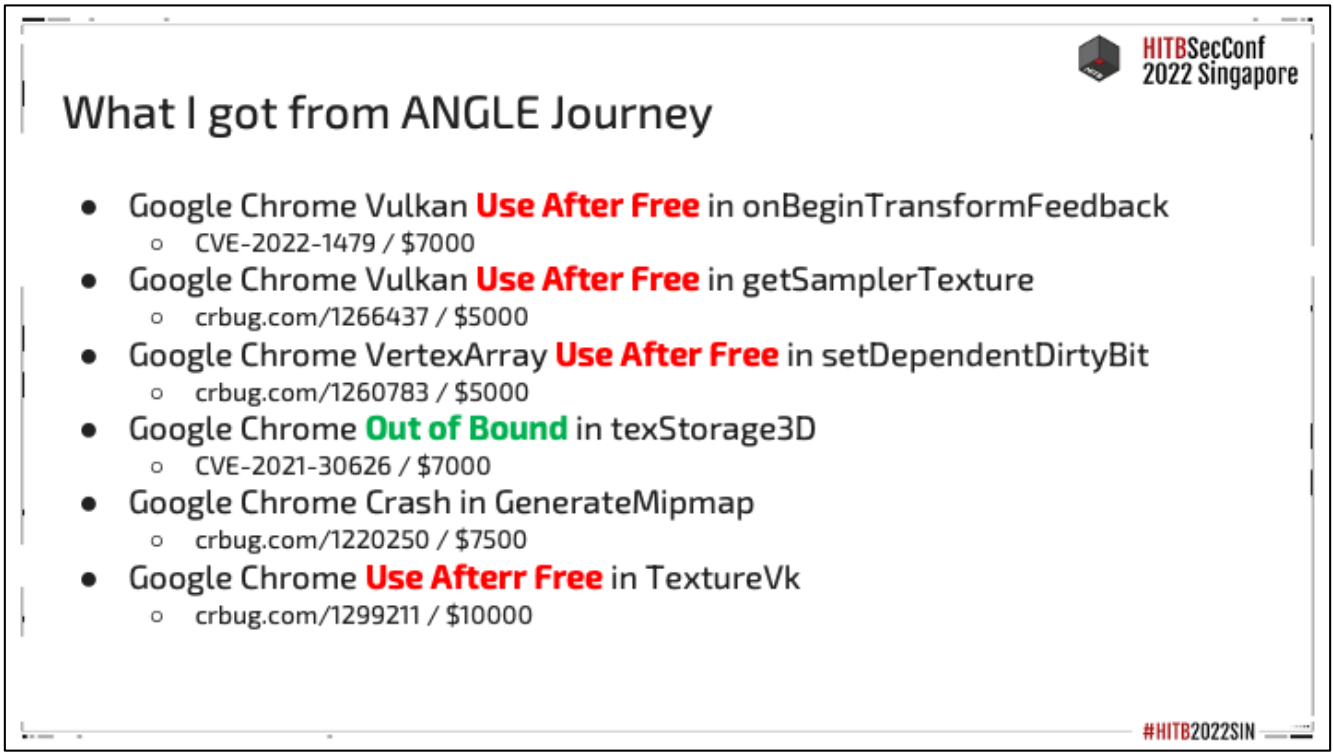


Also, I have traveled(트뤠벌드) to some countries(컨츄리스) around the world for conference speaking.

These are what I talked about before the covid issue.

Um…. Not covid issues have been resolved yet, but

This time, Singapore will be added to this list (디스 타임, 싱가포르 윌 비 에디드 투 디스 리스트.)

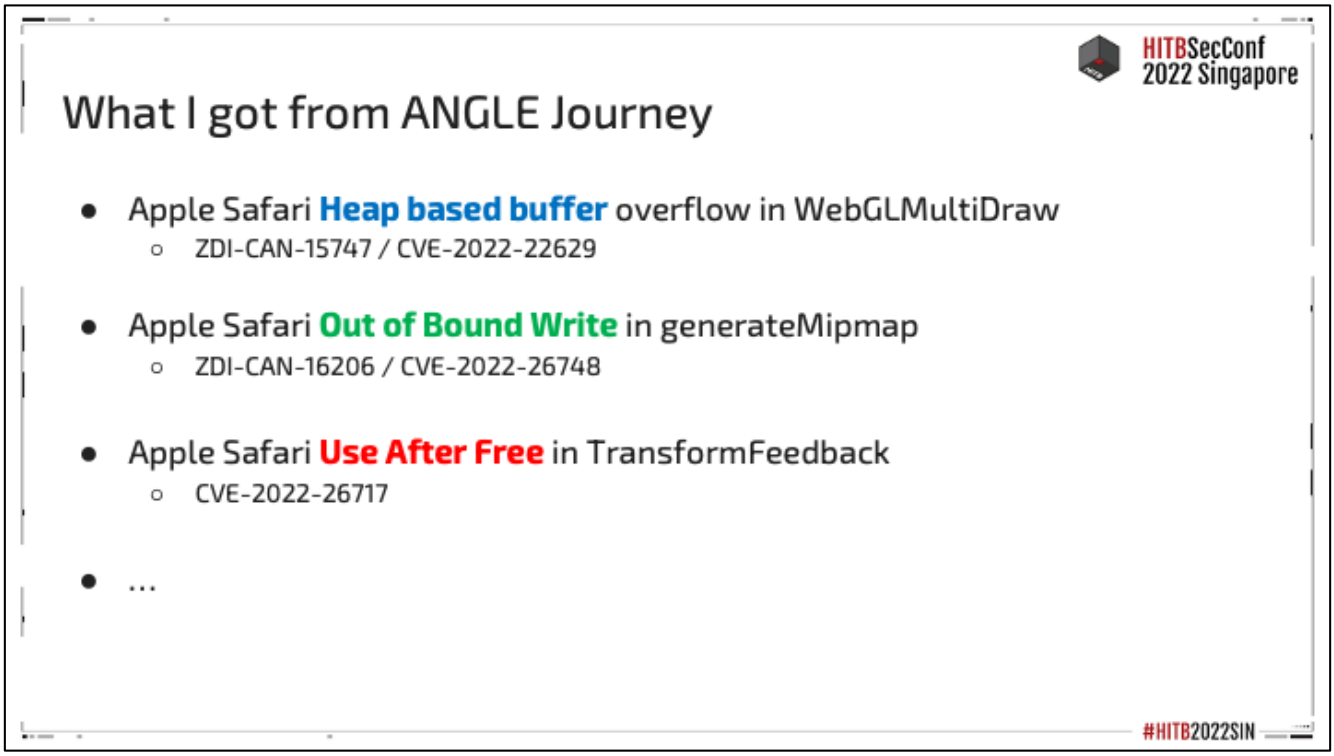


Before we start, I share what I got from / ANGLE journey.(져얼-니)

this bug list is what I reported to chrome.

I had a lot of good experiences with the chrome bug bounty.

If you think your bug is not exploitable, Do report it to a chromium bug bounty program.

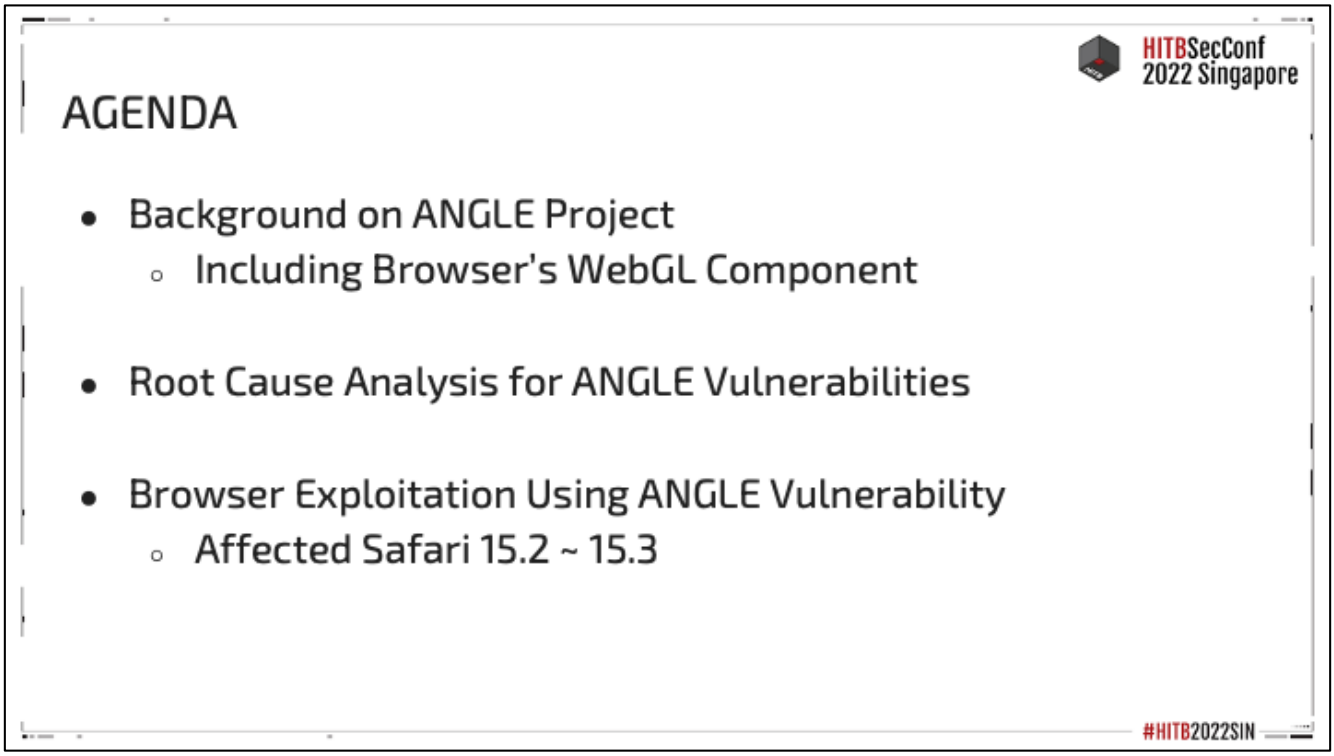


And this list is the bugs I reported to ZDI and Apple.

The first few were reported through ZDI., / And one was reported through Apple.

These are the rewards I got from ANGLE for about one year.

So now, let's get started, Begin with Today Agenda.

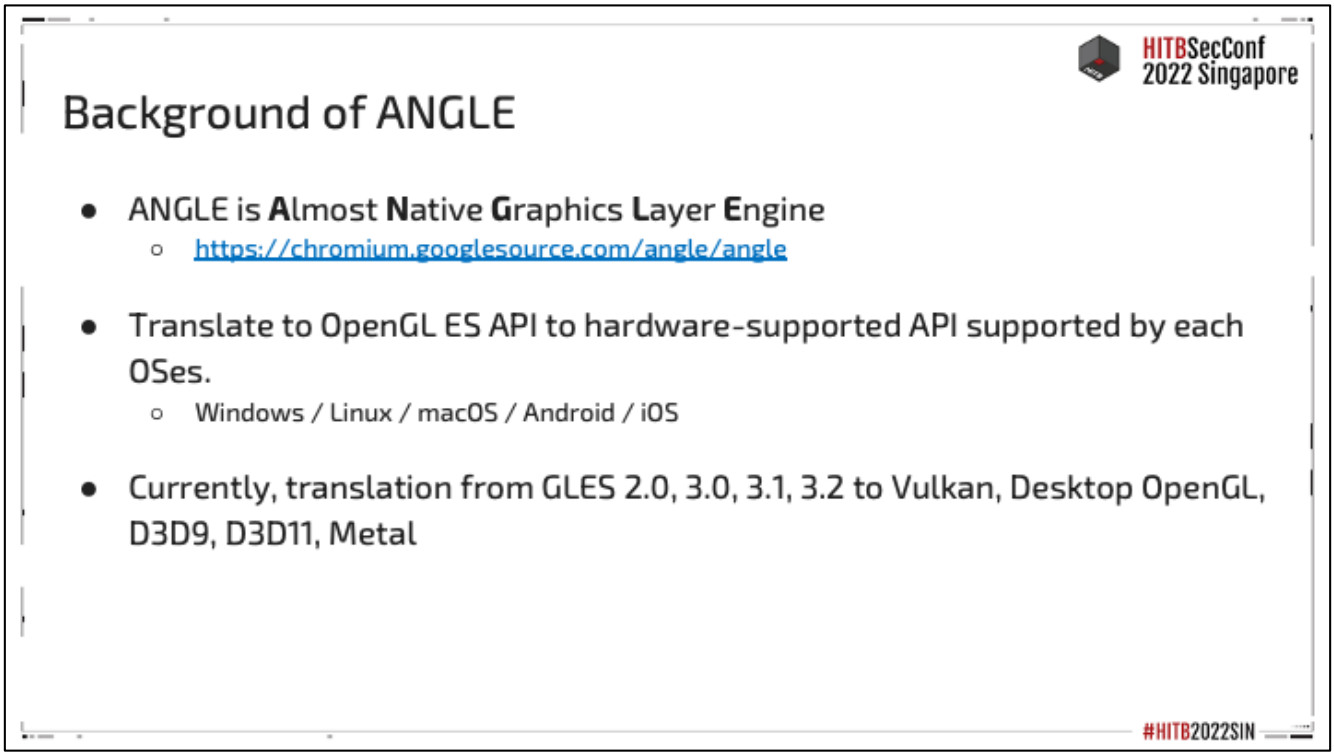


This is Agenda for Today.

We figure out / ANGLE Project and WebGL component.

And Look at the bugs I found in ANGLE project and / Root Cause analysis(어날러시-스) for them.

finally, we're going to look at / how to exploit the Safari browser / with exploitable vulnerability.



So, Let's start with the / definition of an angle.

ANGLE stands for Almost Native Graphics Layer Engine.

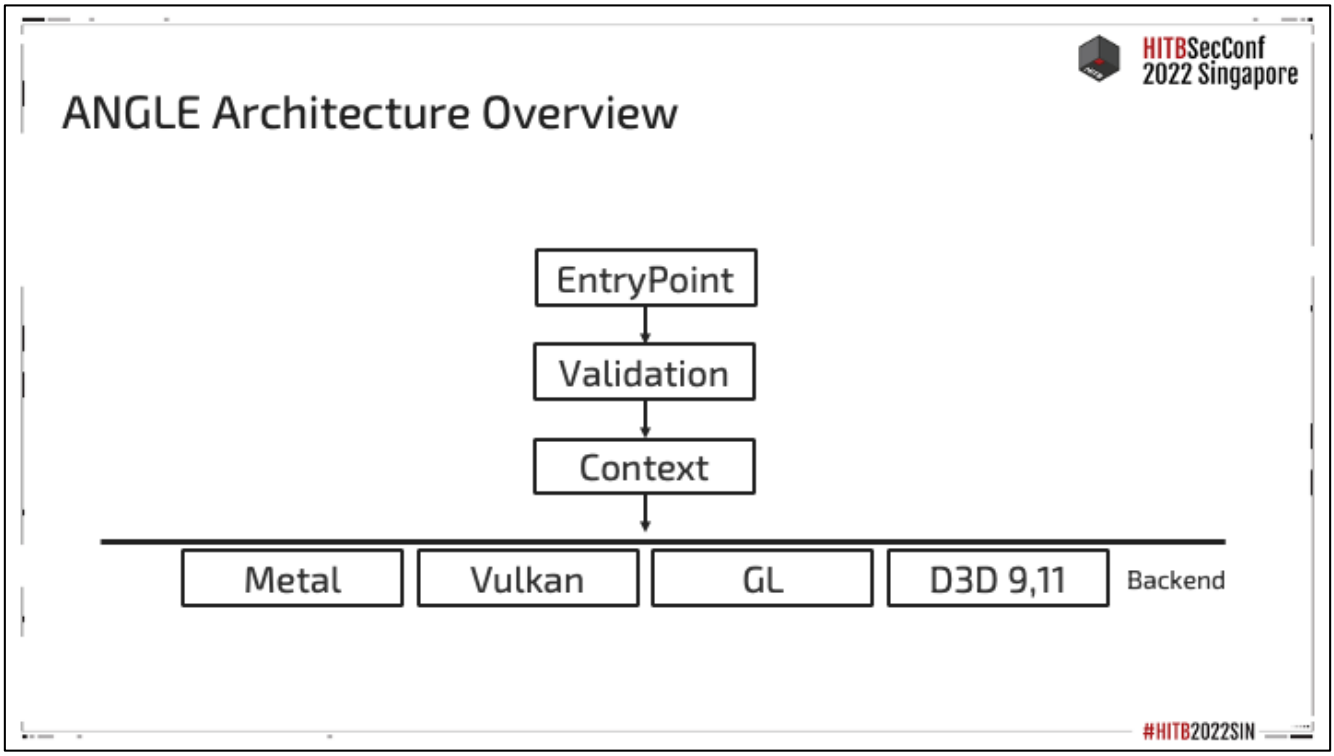
In short, it converts OpenGL APIs to hardware-level APIs supported by each OS.

Its translation(트랜스레이션) from GLES 2.0, 3.0, 3.1(쓰리포인트원), and 3.2(쓰리포인투) to Vulkan, Desktop OpenGL, D3D9(direct 3d 나인) and D3D11 일레븐, Metal backends.

“Metal” sounds a little bit strange. (메탈 사운즈 어 리틀 빗 스트렌쥐) Metal is for mac and iOS backends. as you know, DirectX is for Windows.

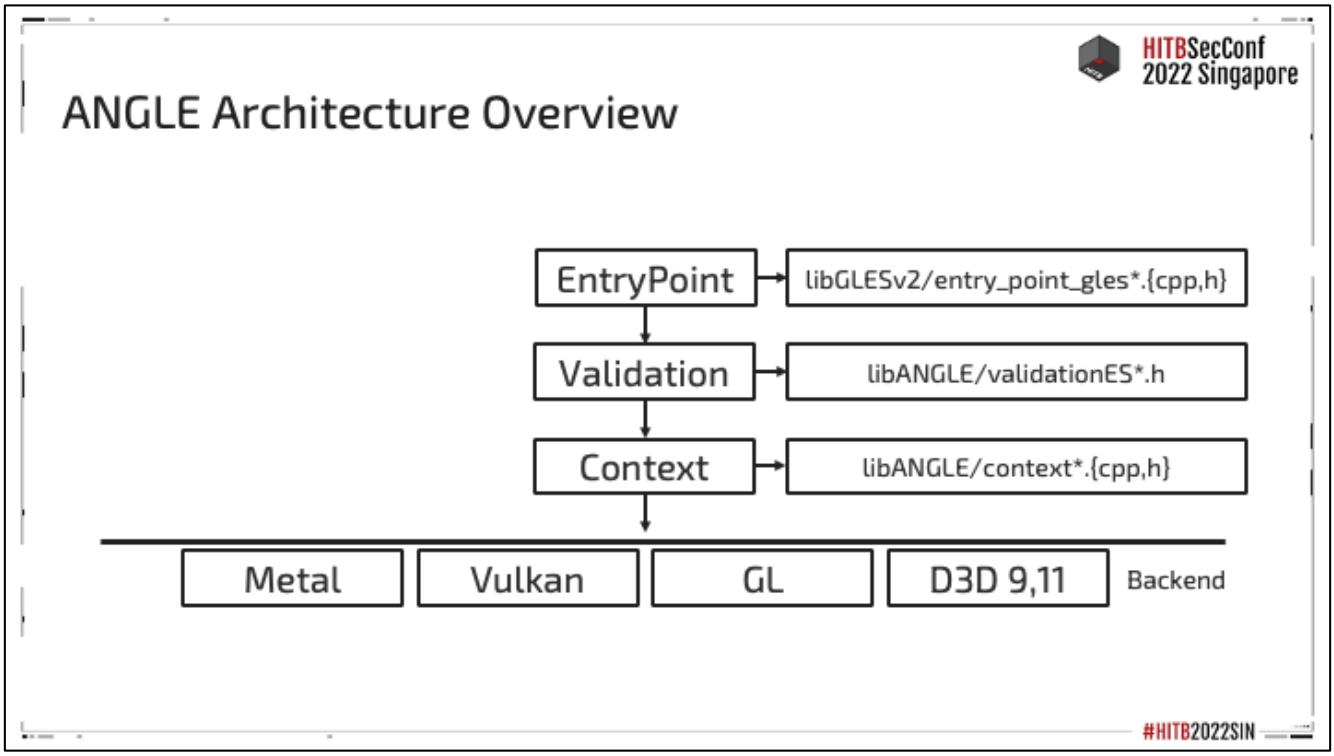
In one sentence, we can say that Angle is a wrapper around OpenGLES.

So, Next, Looking for Angle Architecture. (얼키텍처)



This figure is a simplified representation(심플리파잇 리프리젠테이션) of the ANGLE architecture.

For better understanding,(포 베러 언더스탠딩) I will explain the call sequence of DrawArrays method, which is often used in ANGLE.

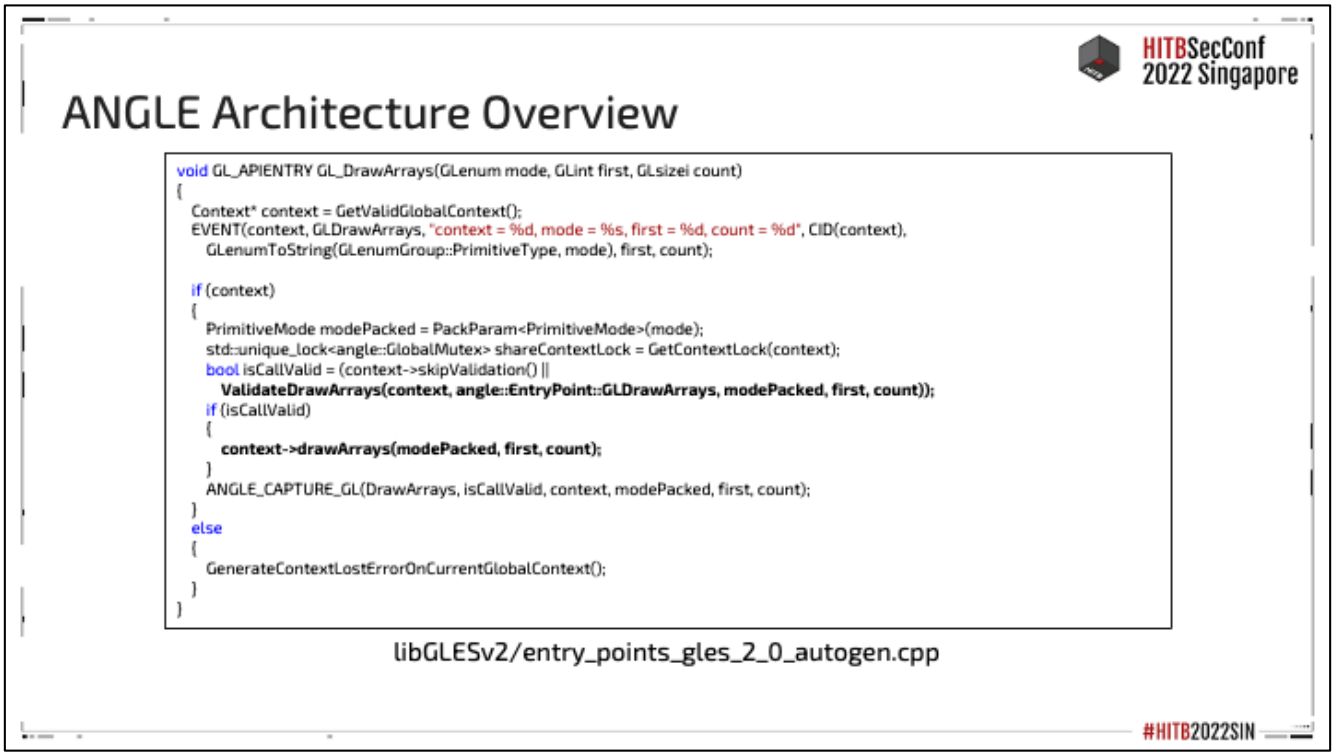


This figure is the file where the DrawArrays method is defined.

Entry points include the Validation and Context part

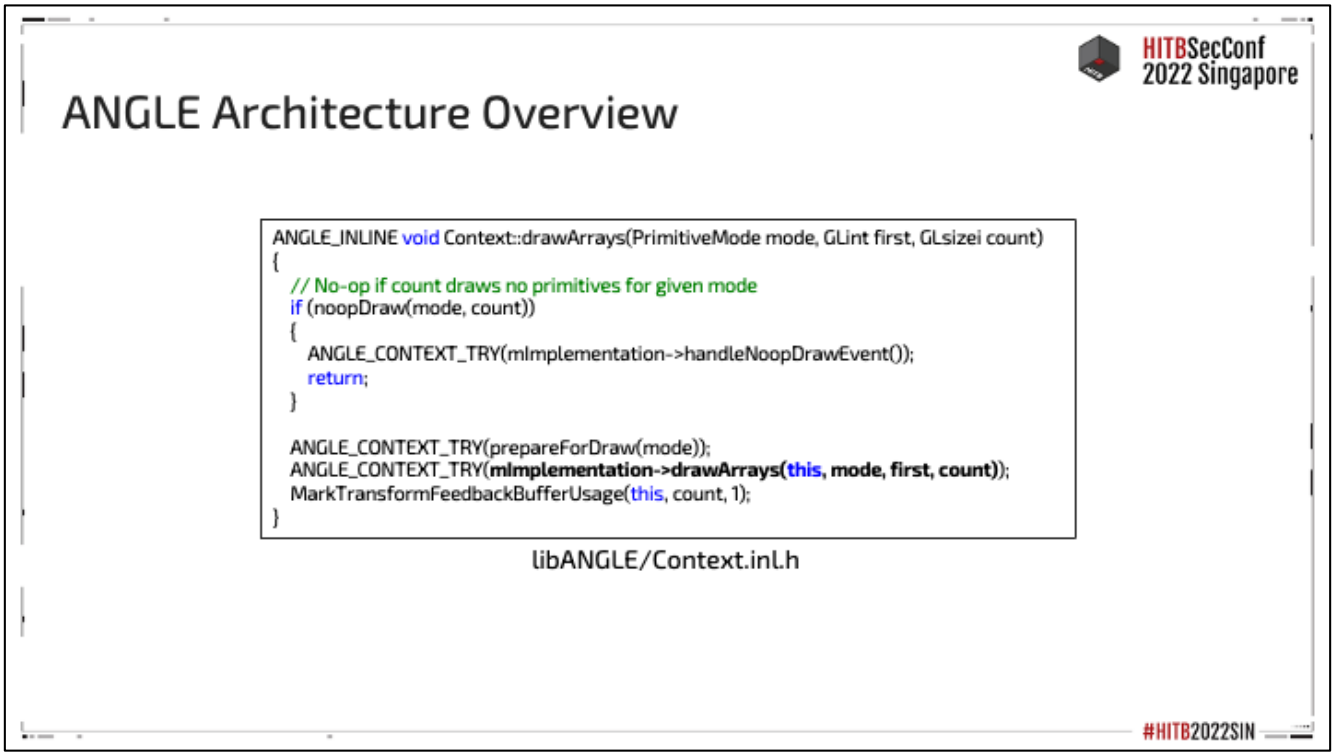
In Validation part,, verifies(베리파이스) the arguments and / types of each function.

the Context part calls the actual(액츄얼) function and converts it / for each OS backend.



This figure is an EntryPoint of DrawArrays.

If you look at this, you can see that / there are validate and context calls, right?



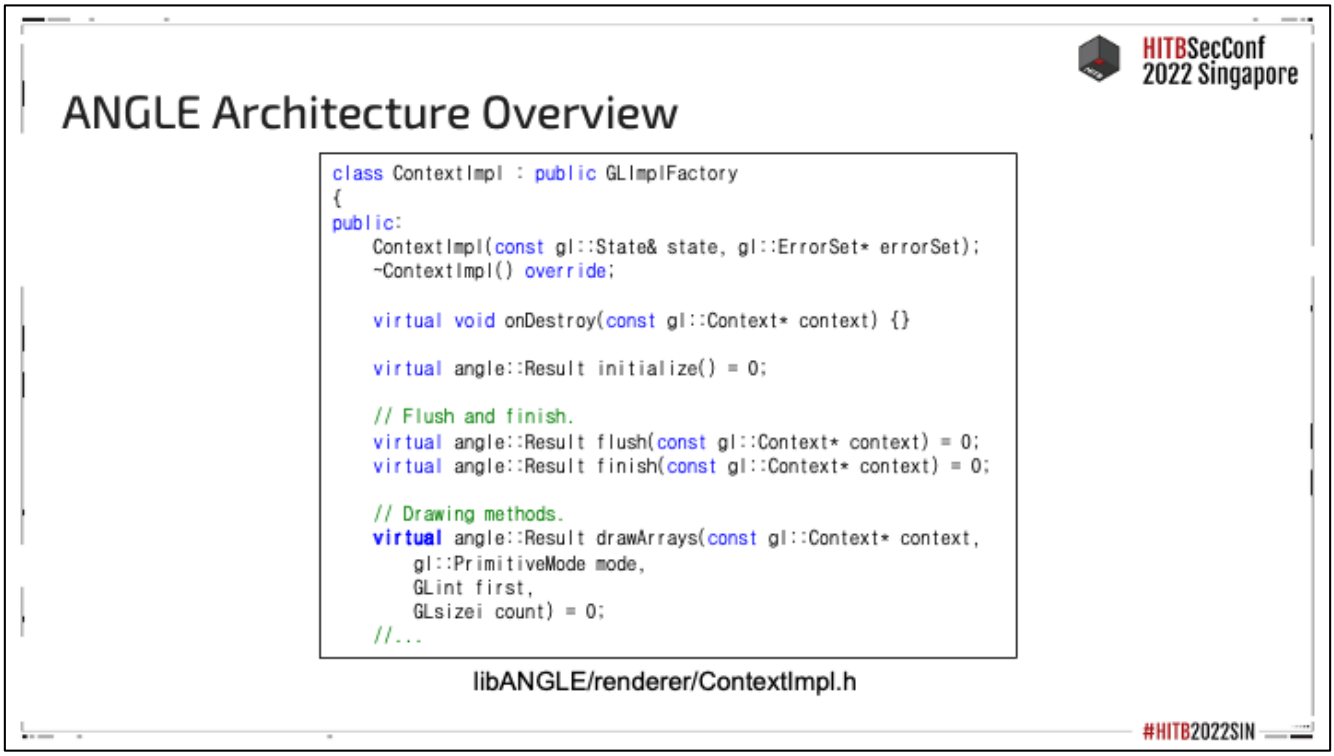
If we pass the Validate part, we reach this code.

Here, call to drawArrays implemented(임플리먼트) for each actual backend.

To implement(임플리멘트) the drawArrays for each backend, ANGLE implements(임플리먼츠) the class method as virtual.

And inherits it from / the backend implementation(임플리멘테이션) class.

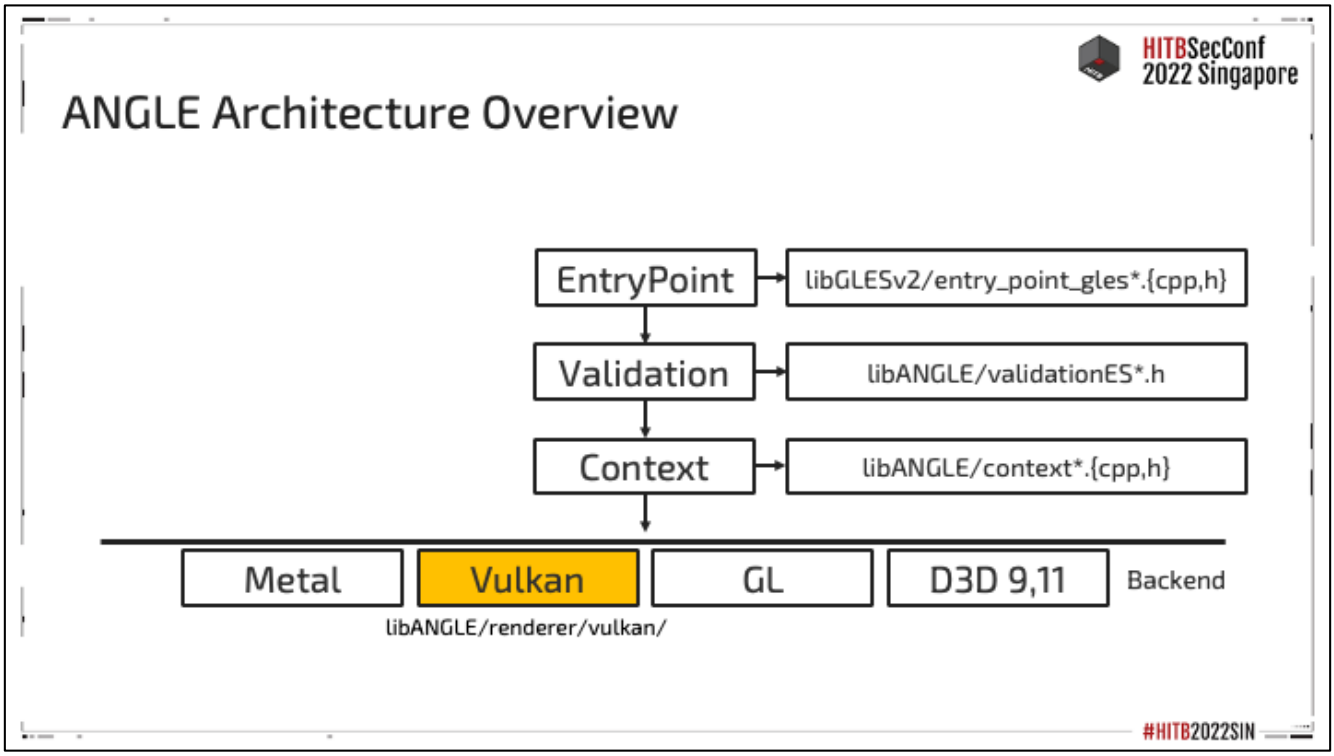
Next, let's check the mImplementation class.



look at the code, drawArrays is defined as a virtual.

Inherit the ContextImpl class from each backend and make the actual drawArrays method.

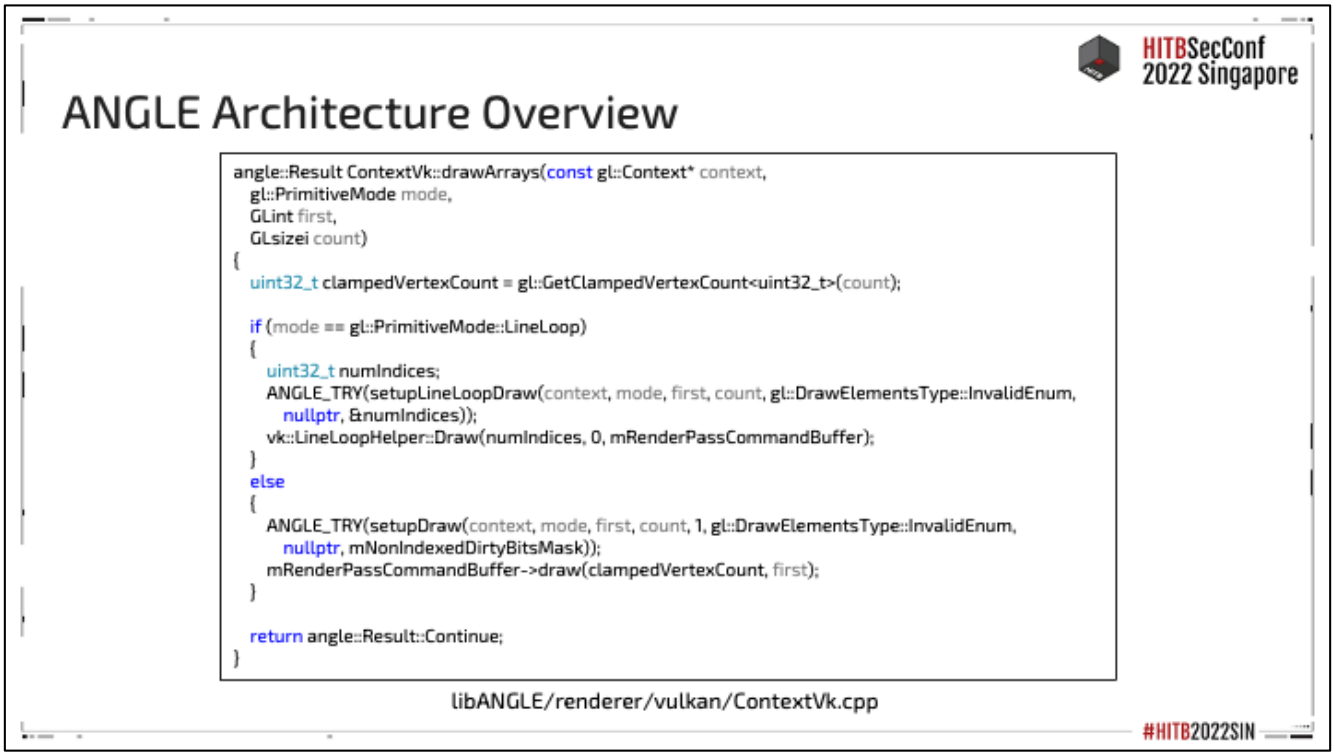
Also, We look at drawArrays in the Vulkan backend as an example.



Just we checked, the ContextImpl class is inherited from metal, vulkan, gl, and d3d 9 and 11 to implement (임플리먼트) actual methods.

Also, other methods / are similarly implemented(시밀럴리 임플리먼트드).

As I said, let's check vulkan’s drawArray method.



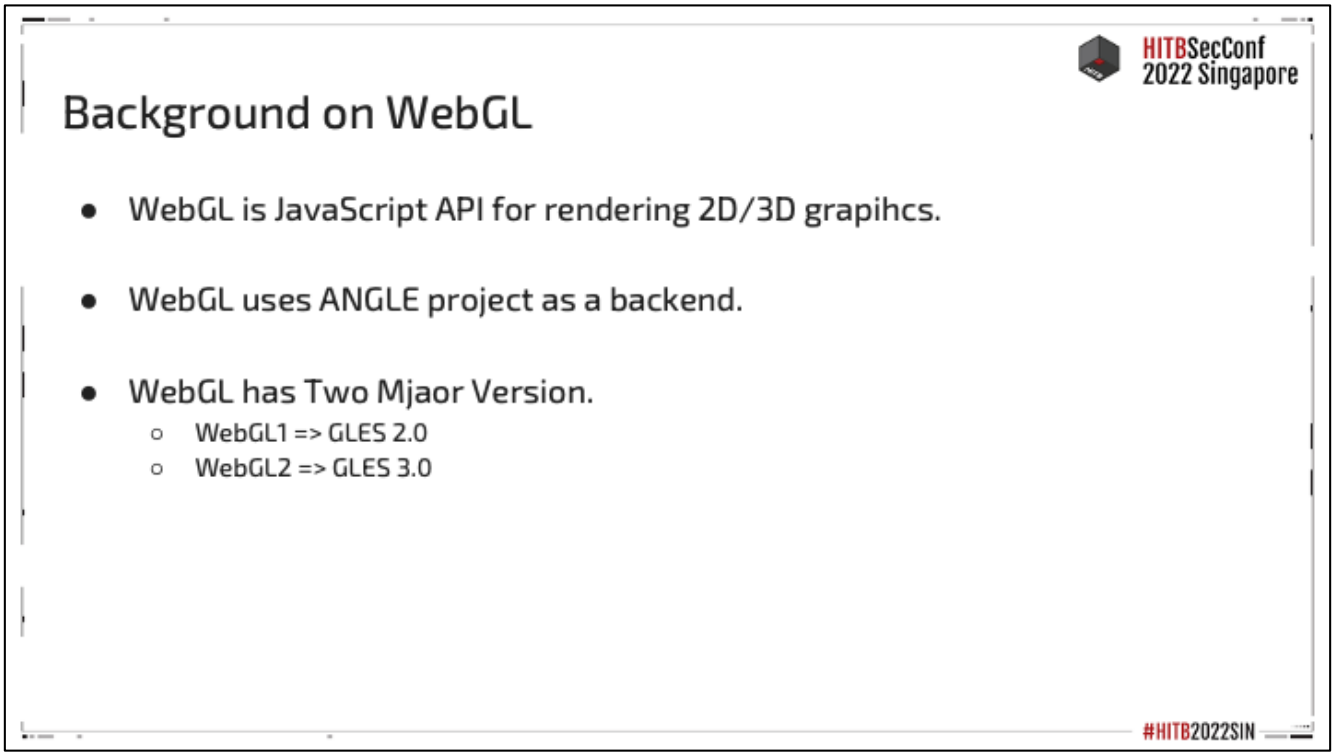
This code is the drawArrays in the Vulkan backend.

Looking at the code, call the setupDraw method and call the draw method.

This code uses the vulkan backend as an example.

so metal and d3d are also implemented(임플리멘티드) with the same structure as this.

So next, let's check the order of calls from WebGL to Entrypoint of ANGLE.

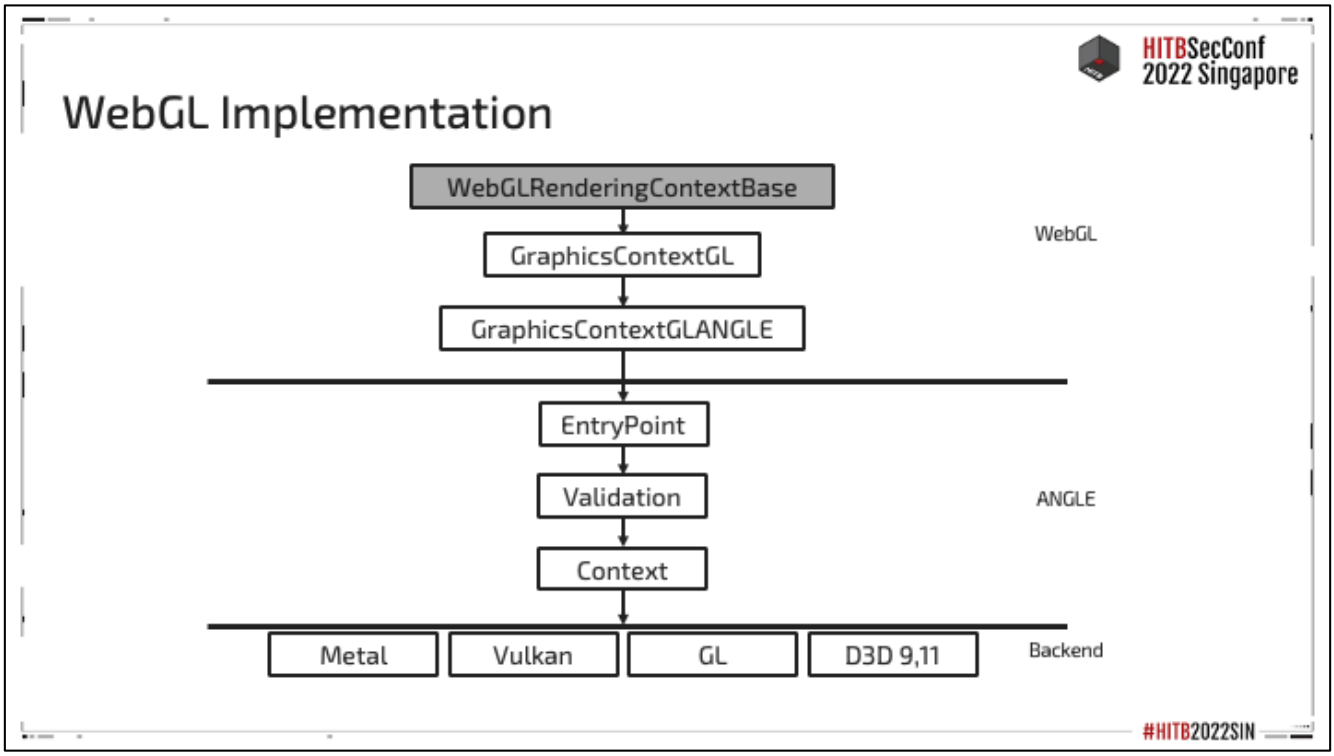


WebGL is a JavaScript API for rendering 2D and 3D graphics within / any compatible(컴패-ㄷ블) web browser without the use of external plugins.

WebGL uses **ANGLE** project as a backend to / support the same level of rendering on multiple platforms.

WebGL has two major versions, WebGL1 and WebGL2. WebGL2 is include WebGL1.

Additionally, the standards for WebGL are defined by the Kronos Group.



It is an architecture that includes WebGL.

ANGLE architecture is the same, only add to the WebGL part.

It is also based on the Webkit browser.

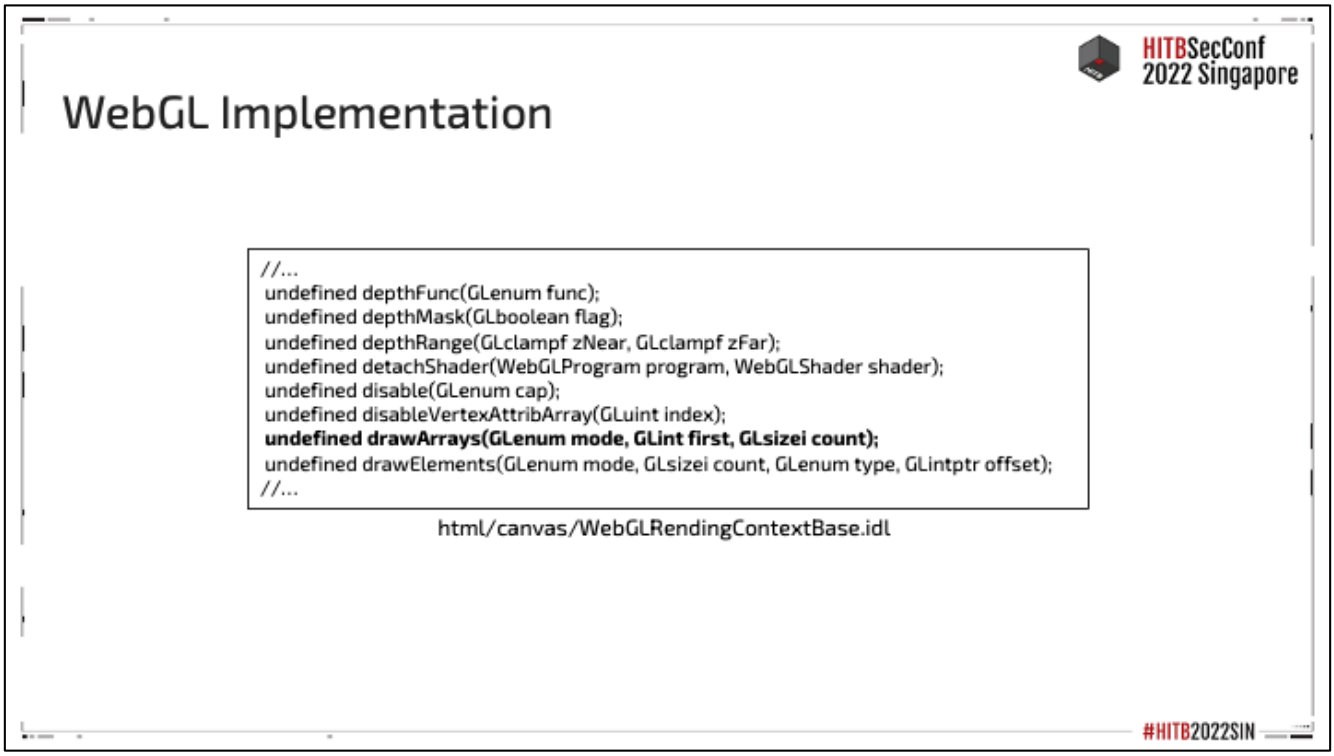
Chrome browser uses an interface named “glesinterface”.

Use that interface to call ANGLE EntryPoint.

However, we will explain based on the safari browser.

Now let’s look at the process of reaching the ANGLE Entrypoint in WebGL.

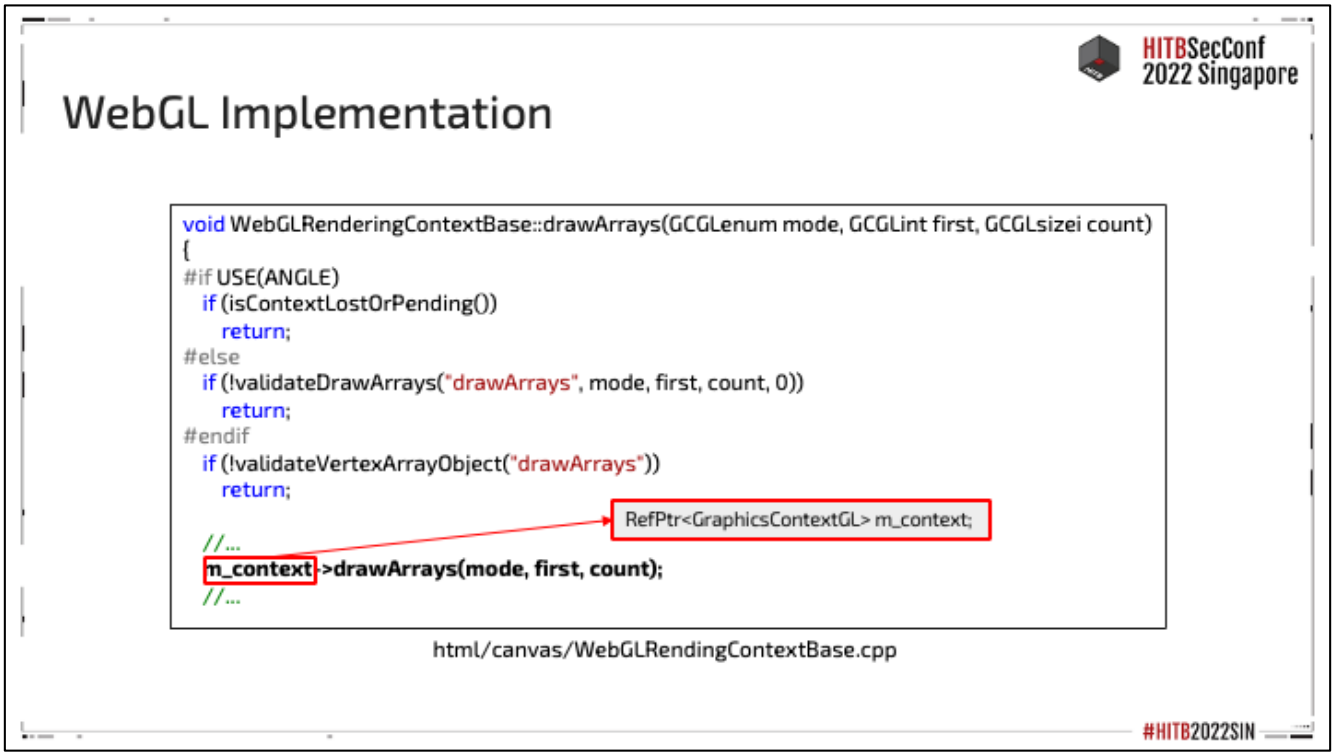
Also, we use drawArrays as an example.



Before that, let's check the IDL where drawArrays is defined.

Usually, methods that users can call / in a web browser are defined in IDL files.

In other words, if we look at IDL, we can check the functions that can be called.

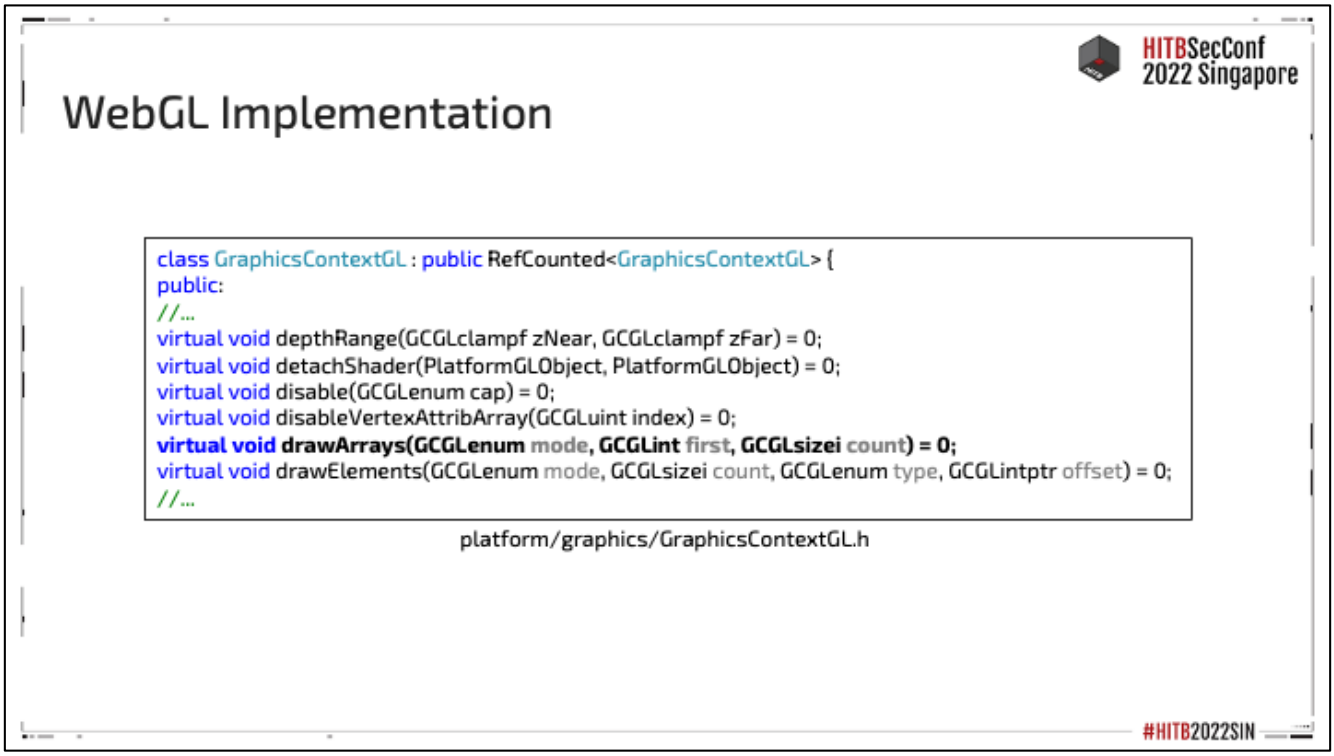


This is the implementation(임플리멘테이션) code of WebGL drawArrays.

There is a couple of validation and after that / the drawArrays of the m_context object is called.

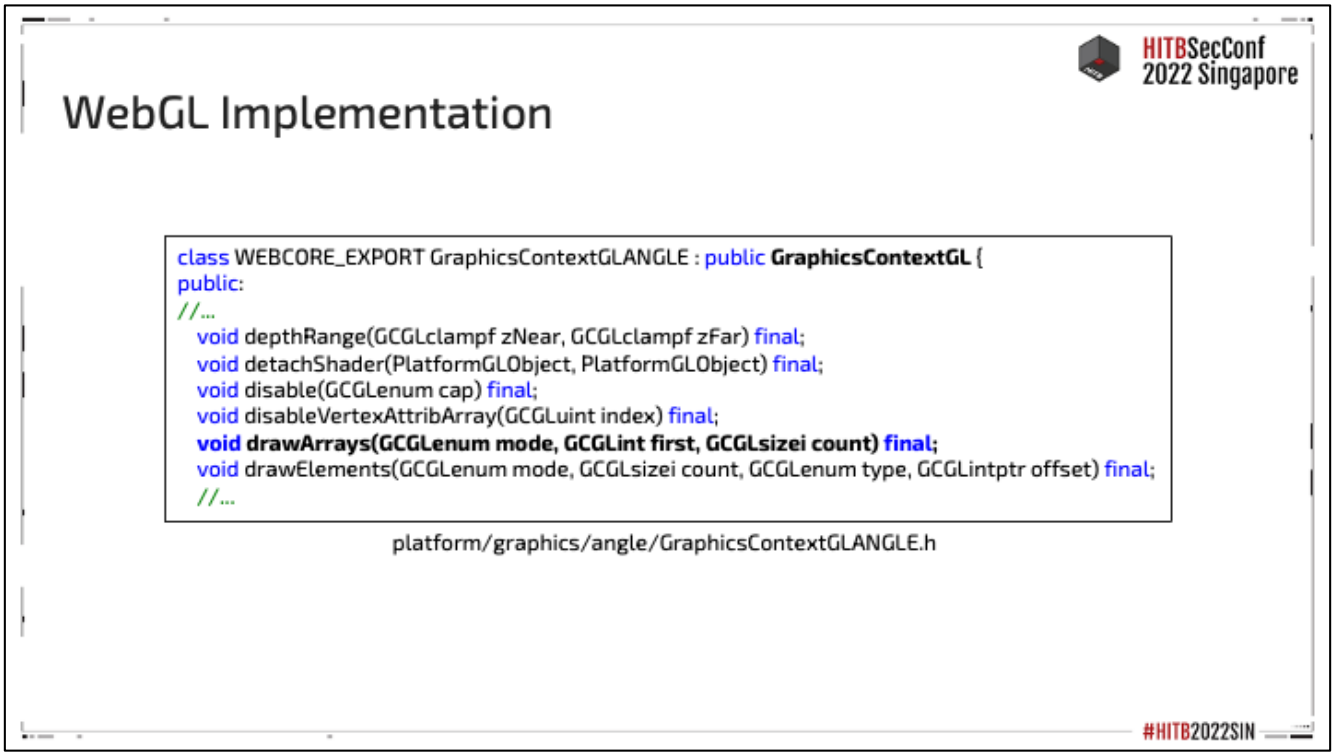
m_context is a GraphicsContextGL object and / is defined in the header file.

Next, we reviewed the GraphicsContextGL object.



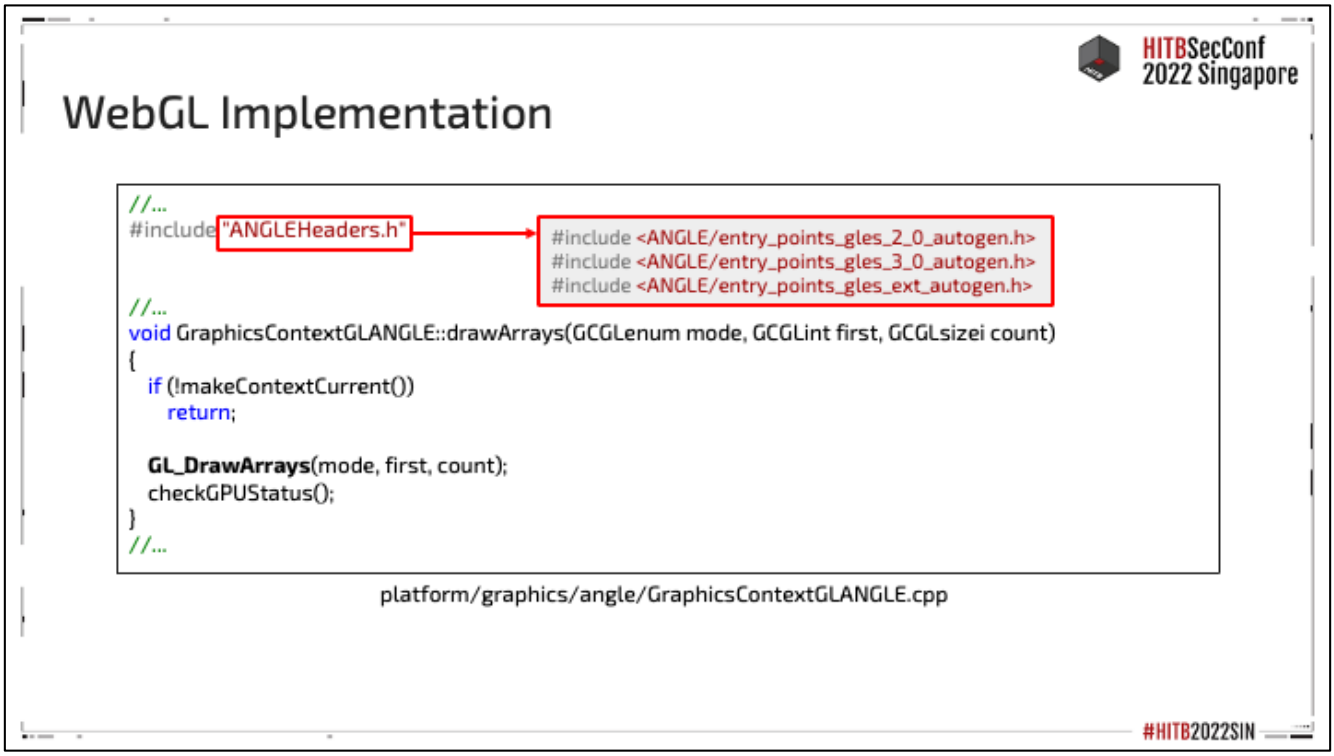
Looking at this class / drawArrays is a virtual method.

So, We need to find a class that inherits(인헤릿츠) this class and / implements(임플리먼츠) the drawArrays method.



This GraphicsContextGLANGLE class inherits GraphicsContextGL class and as we can see, define the drawArrays method.

The following are the drawArrays defined in this class.

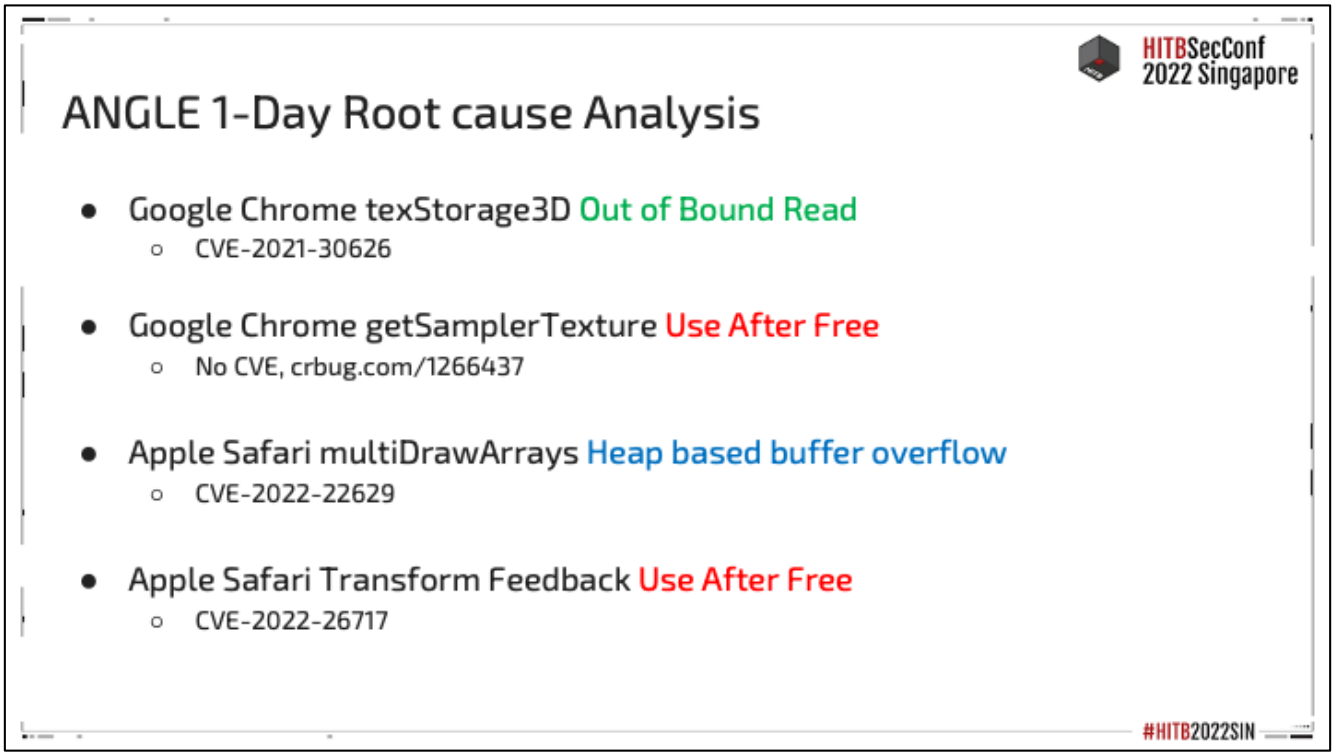


Look at this code, (GL_DrawArrays 가리키며) The GL_DrawArrays is being called.

and as we saw at the beginning of the ANGLE architecture, / this method calls EntryPoint of ANGLE drawArrays.

Okay,, that's it. we learned about ANGLE and WebGL architecture.

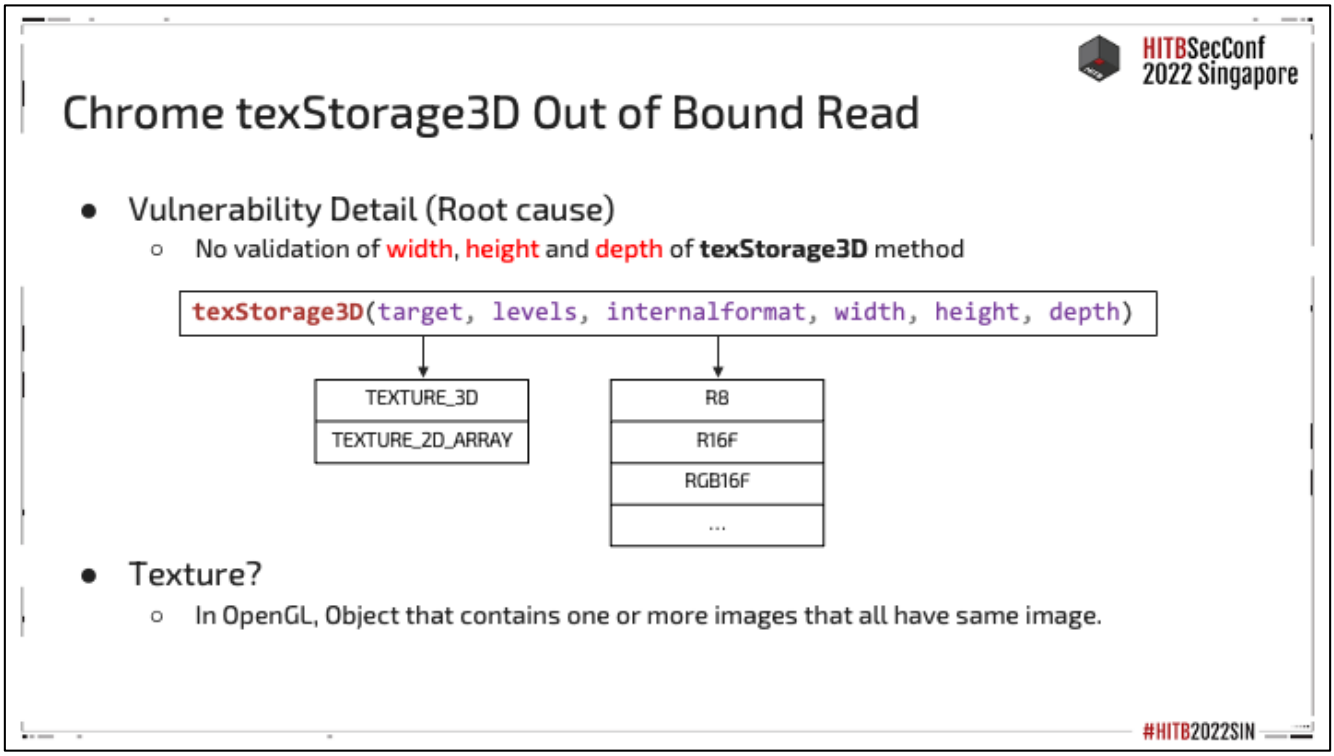
Next, let's take a look at the one-day vulnerability in the ANGLE project.



I will describe the four bug I found.

There are more bugs, but for the sake of time, / I describe the four bugs that I think are interesting.

And we exploit / transform feedback bug in / the exploitation part of this talk.



The first bug to analyze(애널라이즈) is the WebGL bug that I first reported(레포-티드).

Also PoC is so simple, so there was no difficulty in the analysis(어날러시스).

The root cause of this bug is that there is no validation of width, height(하이트) and depth when using the texStorage3D(텍스-스토리지-쓰리디) method.

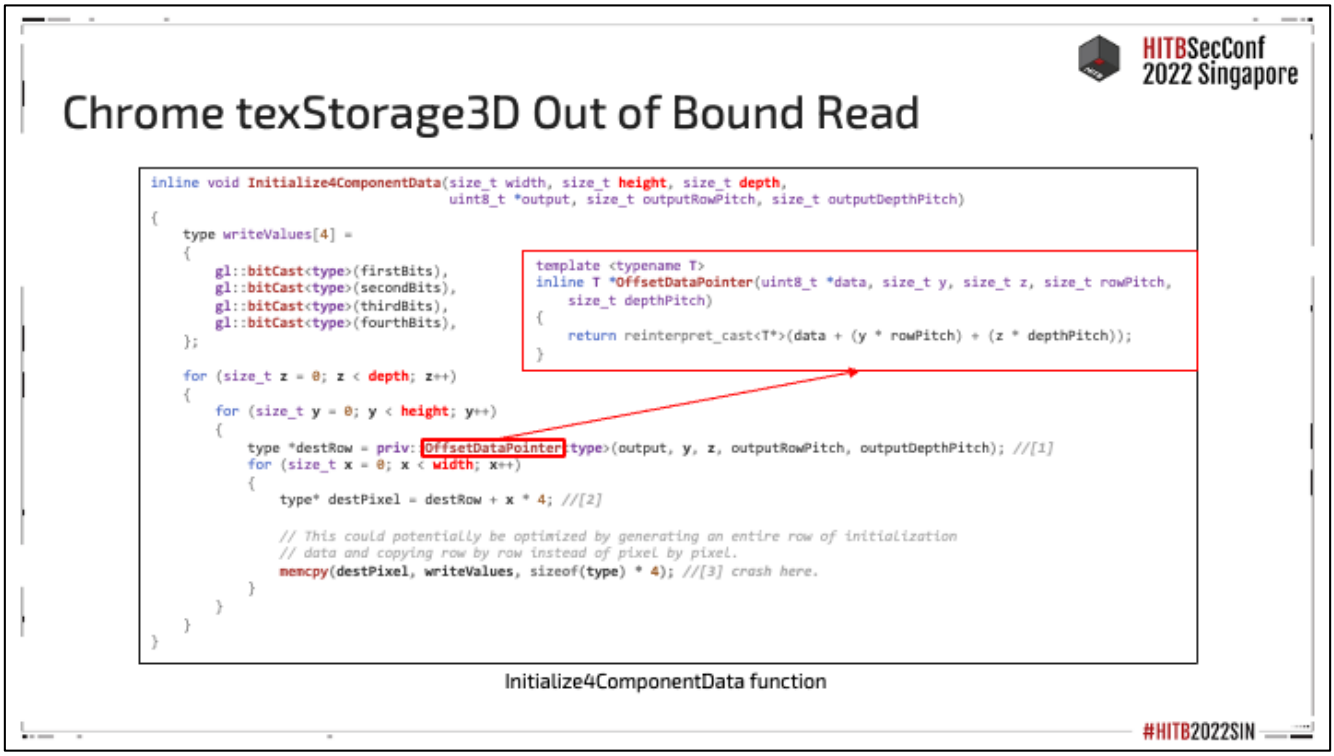
A prototype of texStorage3D is here. (가리키며)

As you can see, all the other arguments are integer types, Except(익!셉트) for “target”, “internal format”.

texStorage3D specifies(스페시파이스) all levels of a three-dimensional(다이멘셔널) texture or two-dimensional array texture.

Also, the texStorage3D method is internally call to `Initialize4ComponentData method. (we covered the next slide.)`

FYI, The texture is an Object that contains / one or more (원 오어 모어) image that all have the same image, also, the texture has a limited size.



A heap based buffer overflow occurs in the comment [3] of this function.

First, get destRow with OffsetDataPointer method in comment [1].

When calling OffsetDataPointer, y and z are used as a(유즈드/ 에즈어) parameter.

Y, and Z are loop counters for depth and height.(헤이츠)

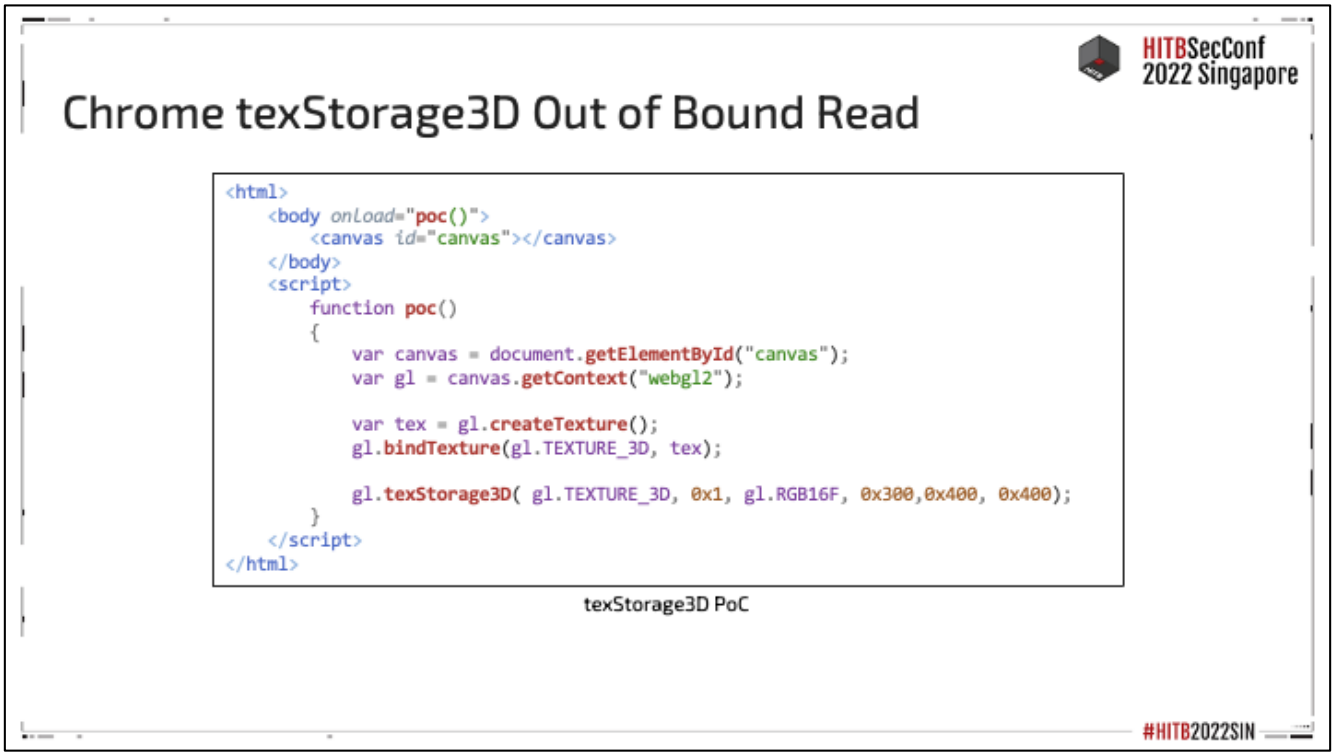
This value is an passed by the user when calling texStorage3D.

As we can see,(해당 함수 가리키며) the OffsetDataPointer gets a memory space to be / accessed(엑!세스드) as y and z based on / data without any verification.

Also, destRow is used to / get destPixel in comment 2.

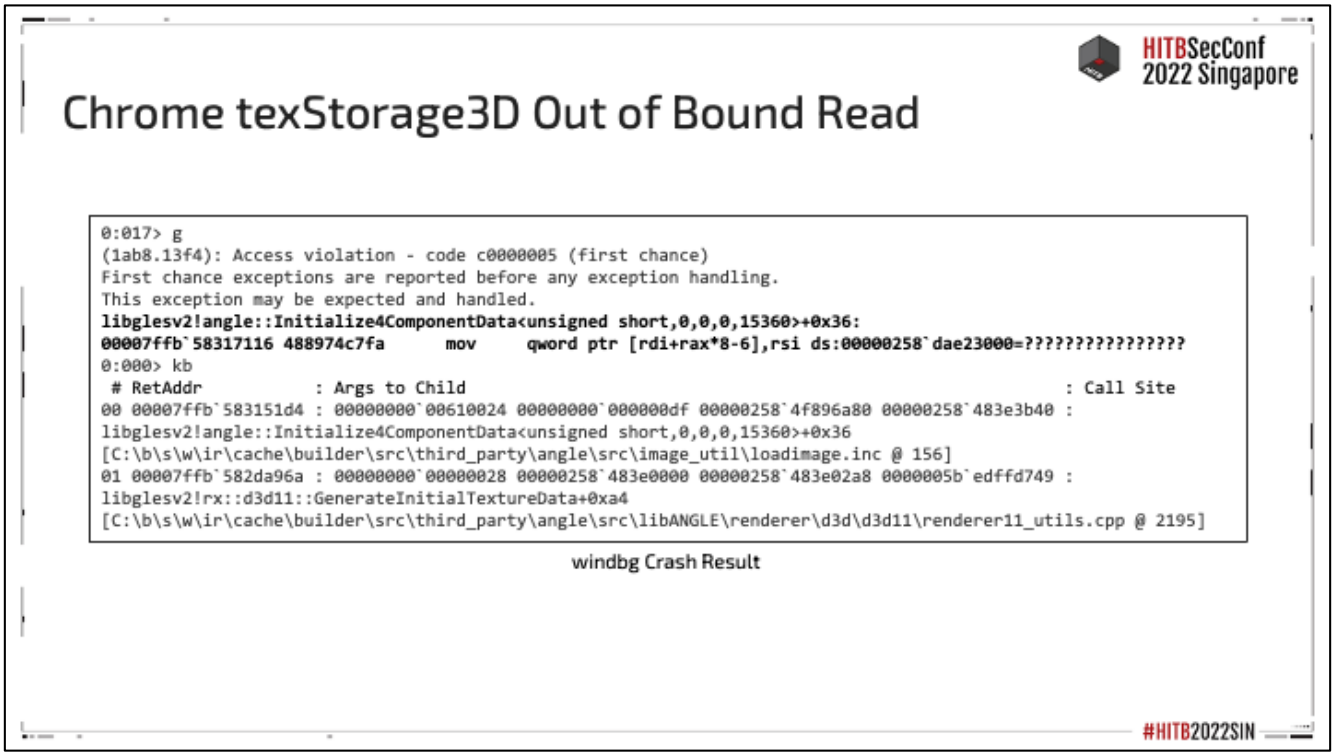
Again, there is no verification here.

If the attacker passes a depth, height, and width that is larger(롸아지) than the current texture size, then out of bound read occurs when trying to access dextPixel in the memcpy.



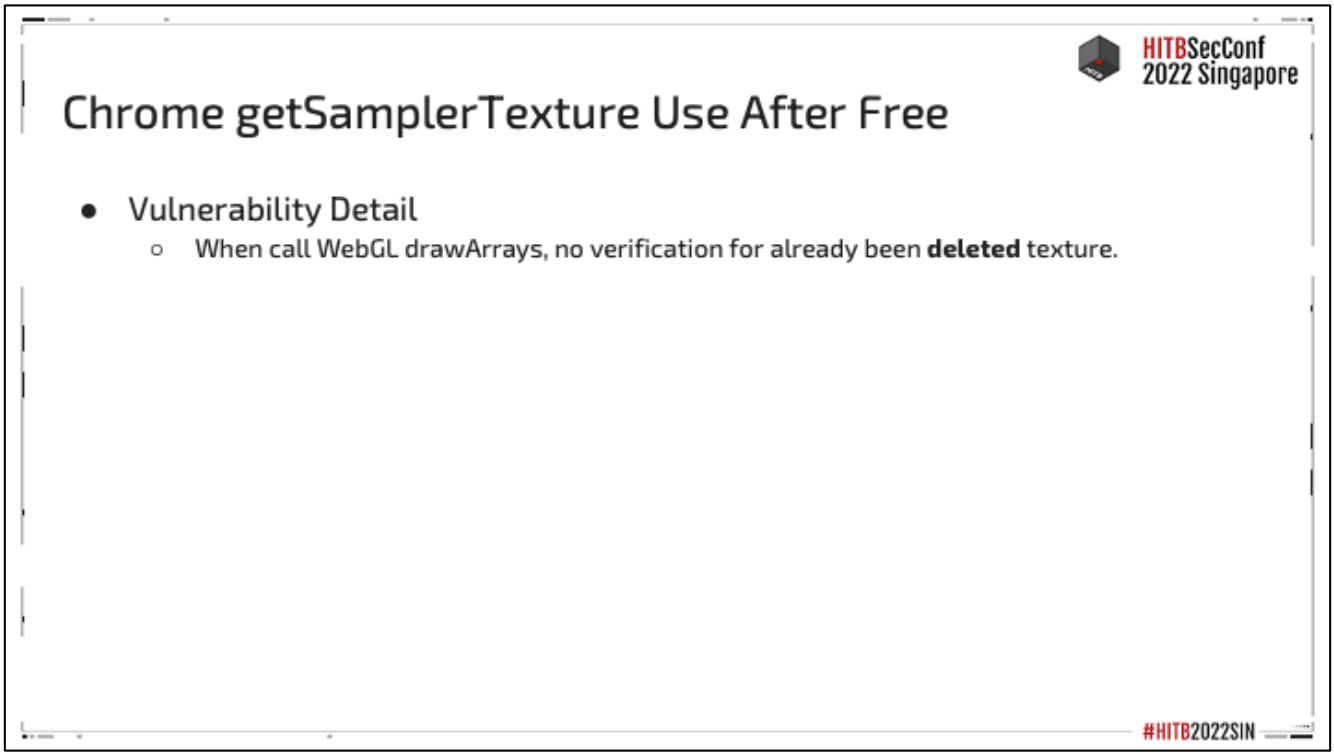
This is the PoC code for the bug.

It is so simple, right?



This result gets after Running PoC with a window debugger. (윈도우디버거)

To get this result, we need to attach the GPU process of the Chrome browser. Not renderer process.



Next, let's look at / an interesting Use after free bug.

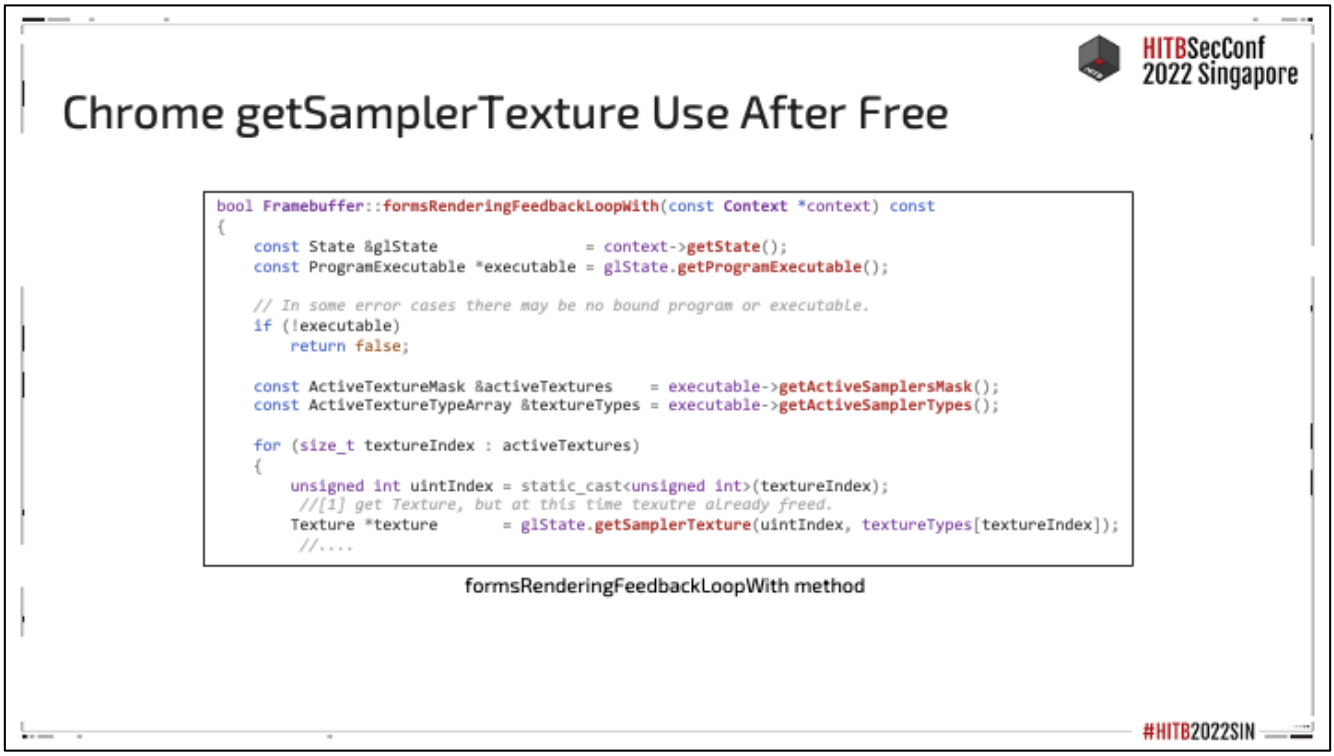
This is a bug that / occurred(어컬드) in the drawArrays method. But that / doesn't mean, it's a problem in the drawArrays. (더즈 낫 민, 잇츠 프러블럼 인더 드로우어레이즈)

drawArrays did several(세브럴) verifications before drawing. / when do this, there was a problem with the object being verified.(배러파이) (drawArrays 메소드가 drawing 전, 여러 검증을 진행 합니다. 이 때 검증 대상이 되는 객체 문제가 발생했습니다.)

The bug type is Use After Free,

This is caused(커어ㅈ) by not validating / already removed Texture object in the drawArrays verification process.

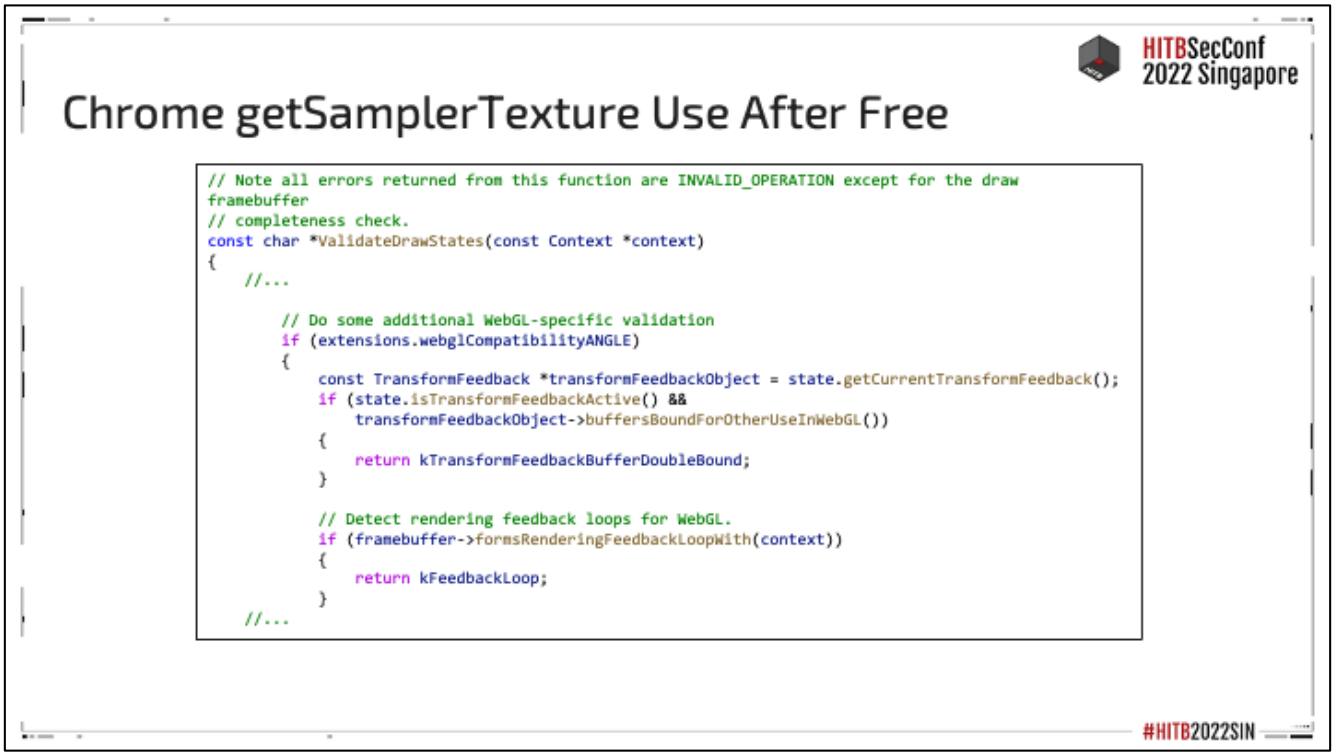
Also, this bug pattern helped / to find an exploitable vulnerability in Safari.



This “가리키며” function is called internally when drawArrays is called.

If you look at the comment [1], Use After Free occurs(어컬스) when you use / a texture that has already been removed.

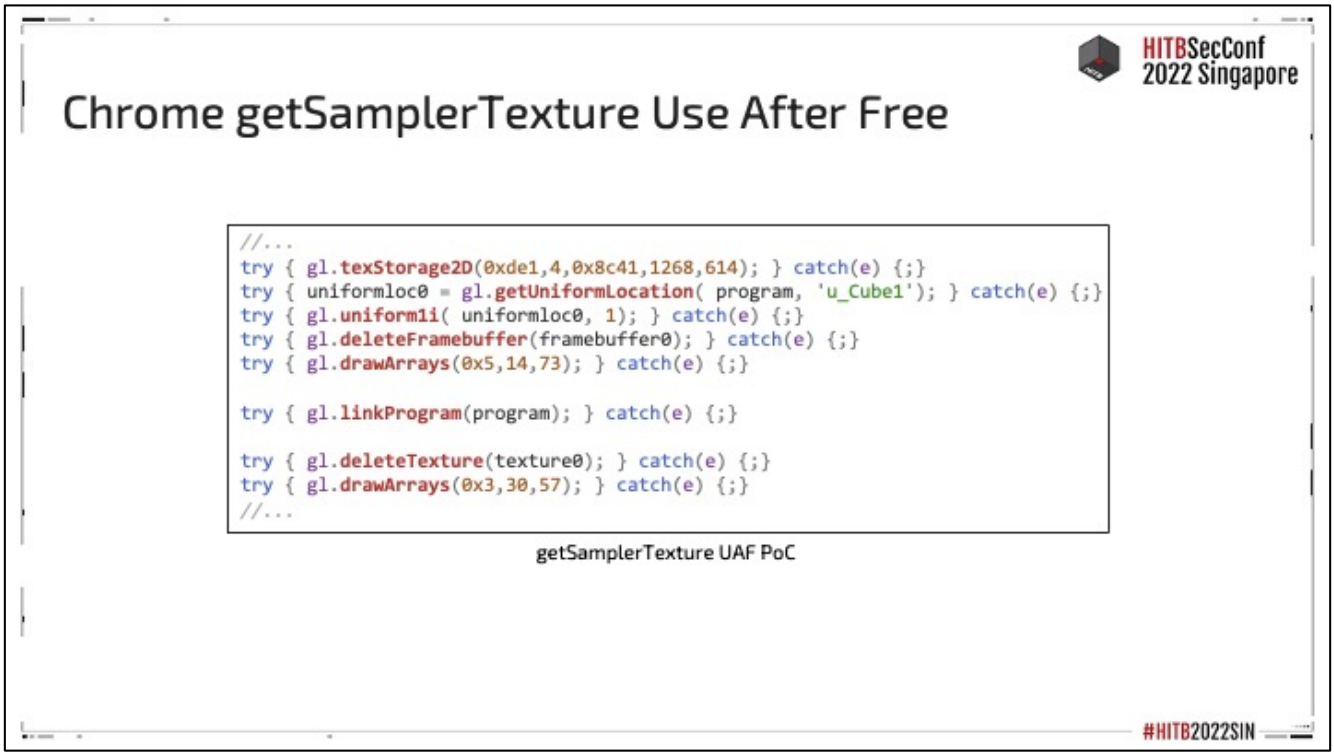
This function is / one of the functions that drawArrays validates several states before drawing.



Looking at the code comments, it is expected(엑스펙티드) to be a function that validate whether a rendering feedback loop exists(이그짓스트) in the framebuffer before drawing.

It seems weird(위얼드) to / mention of texture object in the framebuffer validation function.

But, according to OpenGL standard, / it is specified(스페시파이드) that the Framebuffer Object should be / use a texture or RenderBuffer object as the Rendering Target.

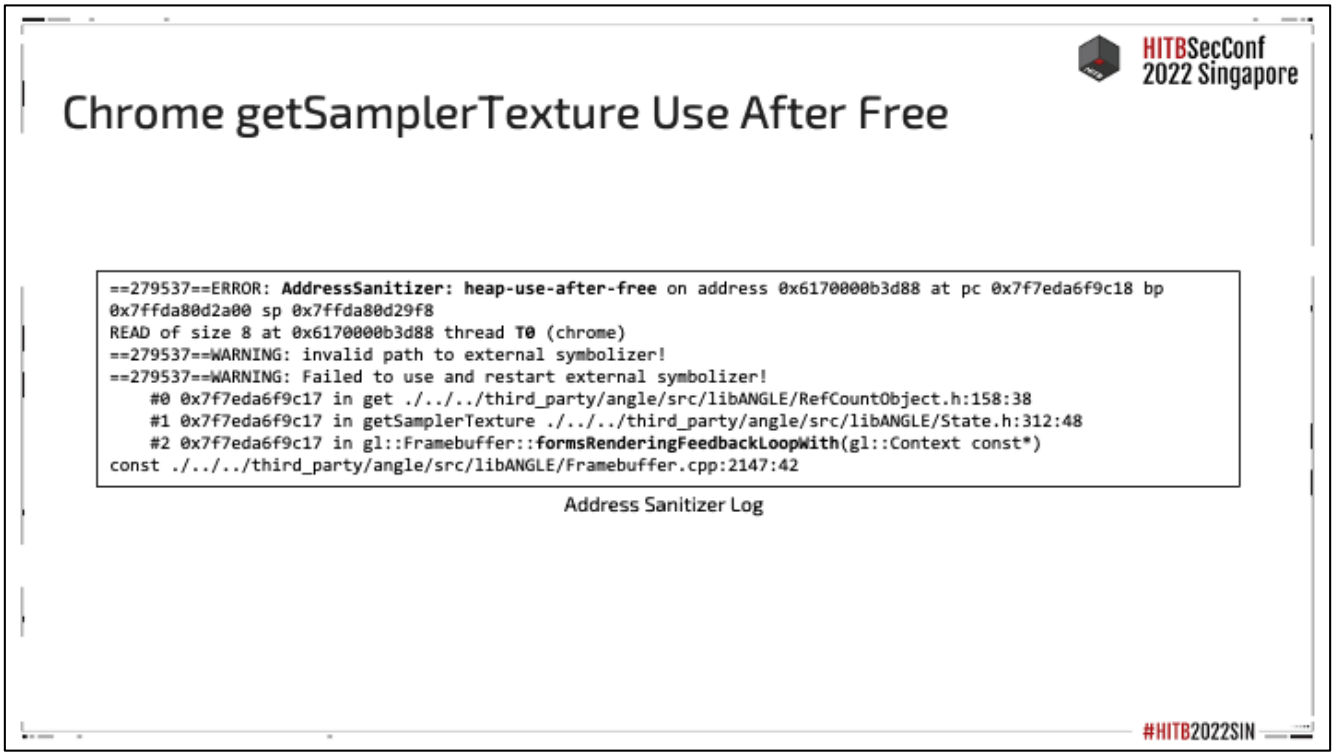


This code is part of the PoC code.

As you can see, to trigger is so simple too.

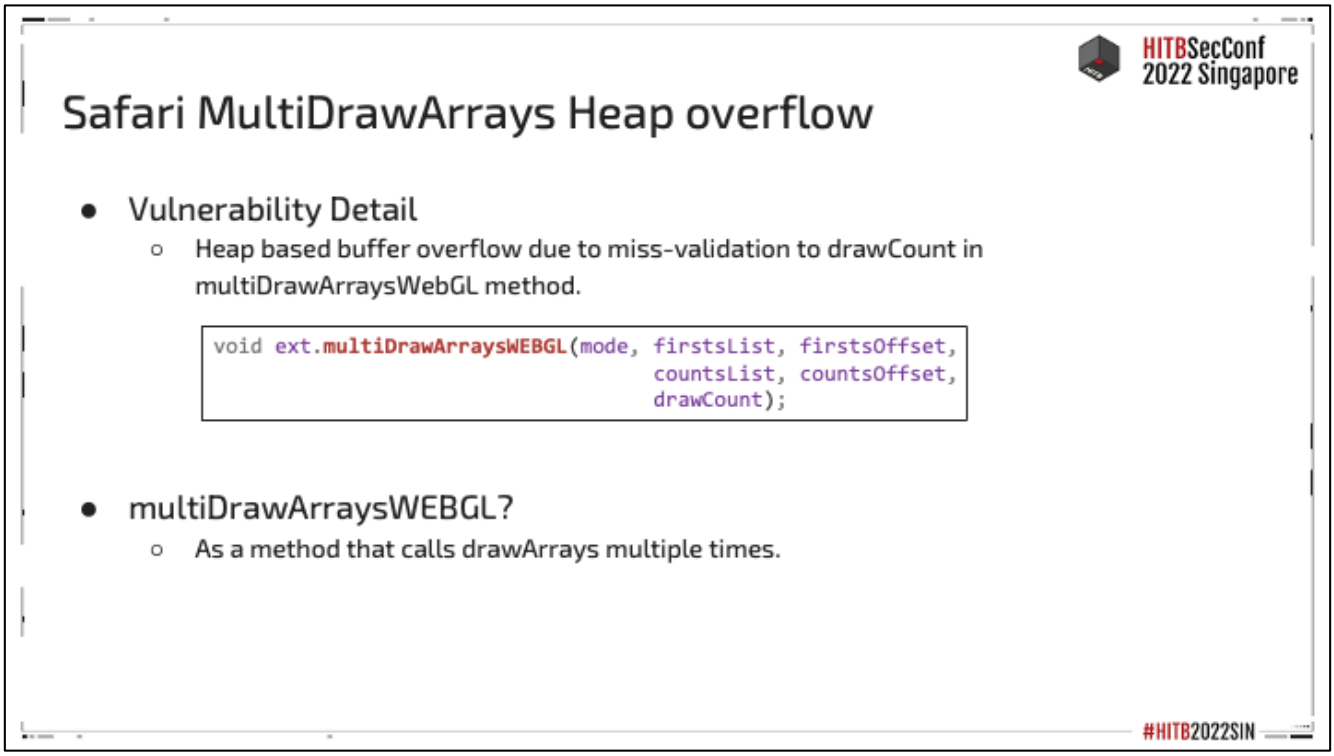
you just need to / remove the bound texture before calling drawArrays.

If you want a completed PoC code, can be found by searching for the issue number or title on crbug.com



This is the result getting with Address Sanitizer.

Next, we move on to Safari vulnerabilities.



As we knew, safari use ANGLE too.

So, when testing ANGLE bug , we should test whether they can be triggered in chrome, safari, and firefox.

This bug can be triggered in Chrome and Safari browser.

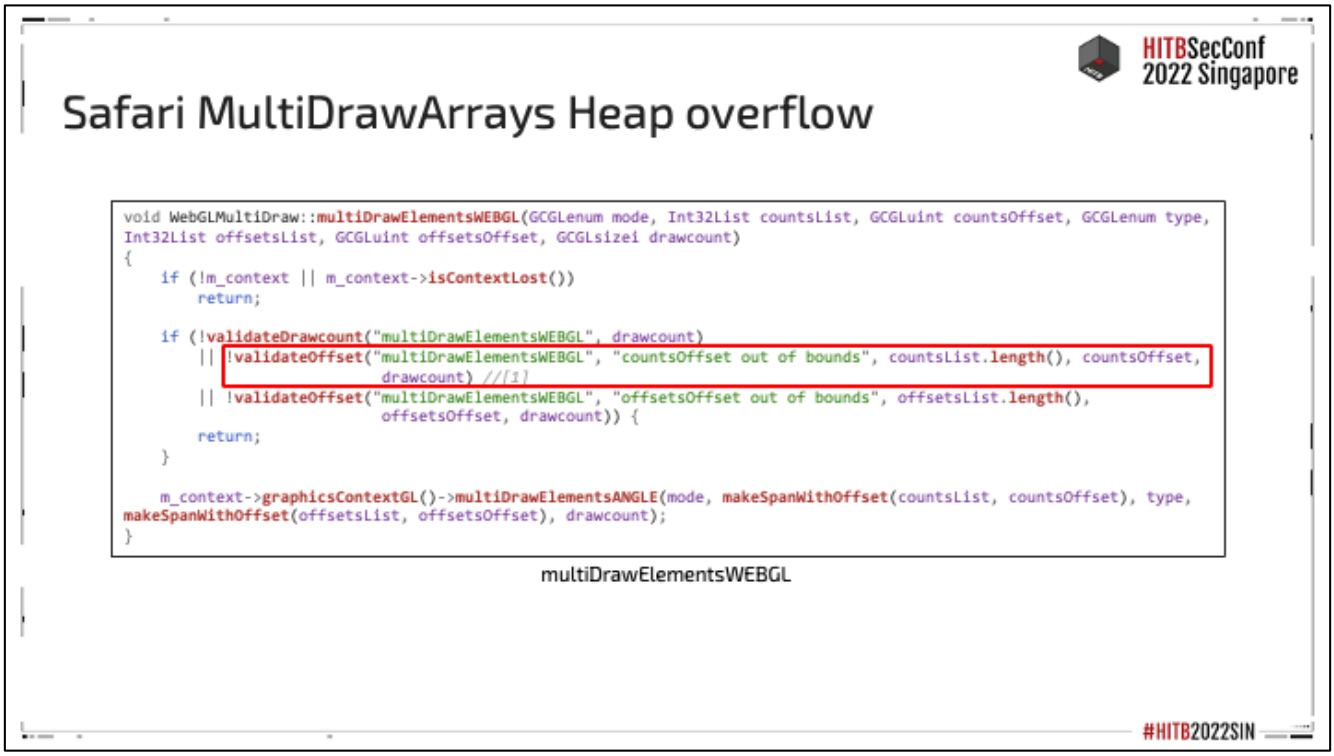
And the bug type is a heap buffer overflow,

caused by insufficient parameter(인써피션트 프아라미럴) validation of this(multiDrawArraysWEebGL 가리키며) method.

Also, the method(가리키며) is a wrapper(랩허) method that calls drawArrays multiple times.

it is not the default WebGL method, So to use this function, we need to enable some extension.

To enable the extension is shown in the PoC part.

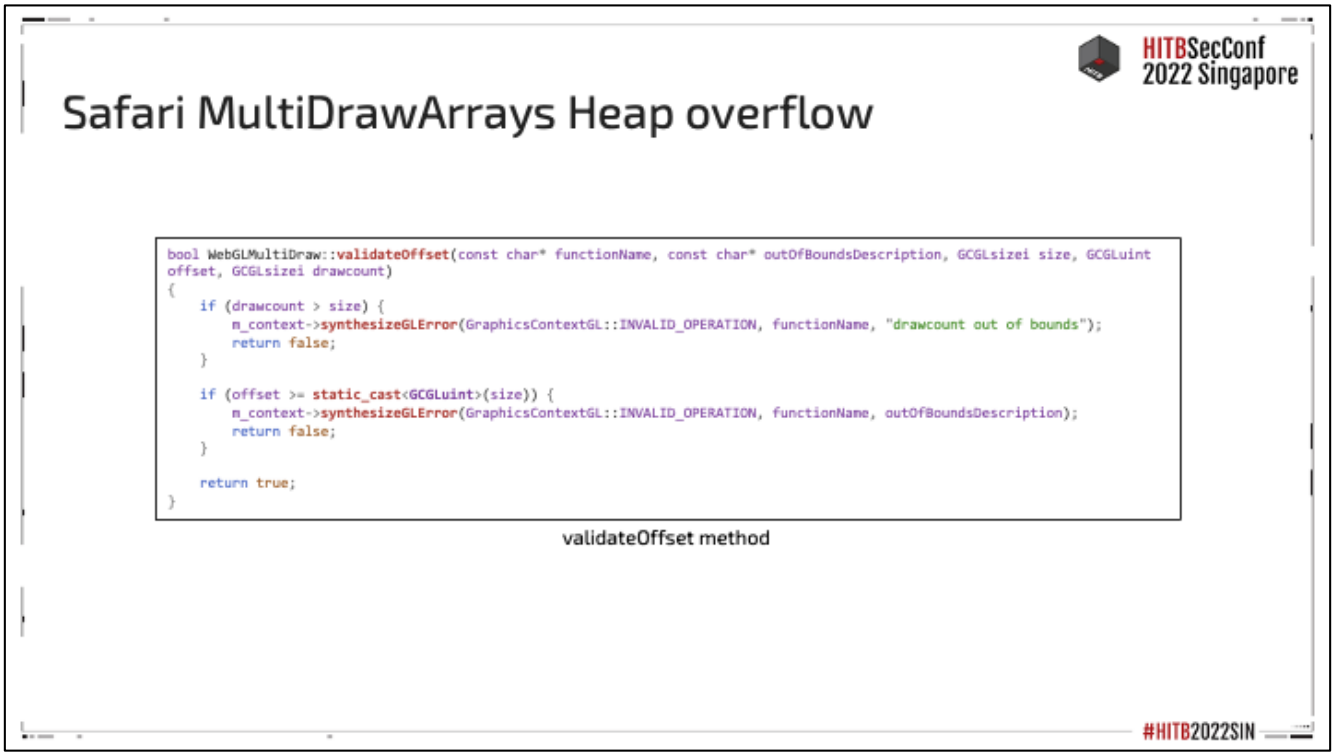


This code is webgl's multiDrawArraysWebGL method.

Already we know, this is not a part of ANGLE. / It is WebGL part in the Safari Browser.

So, The problem is this point.(가리키며) Not enough validation for drawCount.

Let's take a closer look / at the validateOffset.

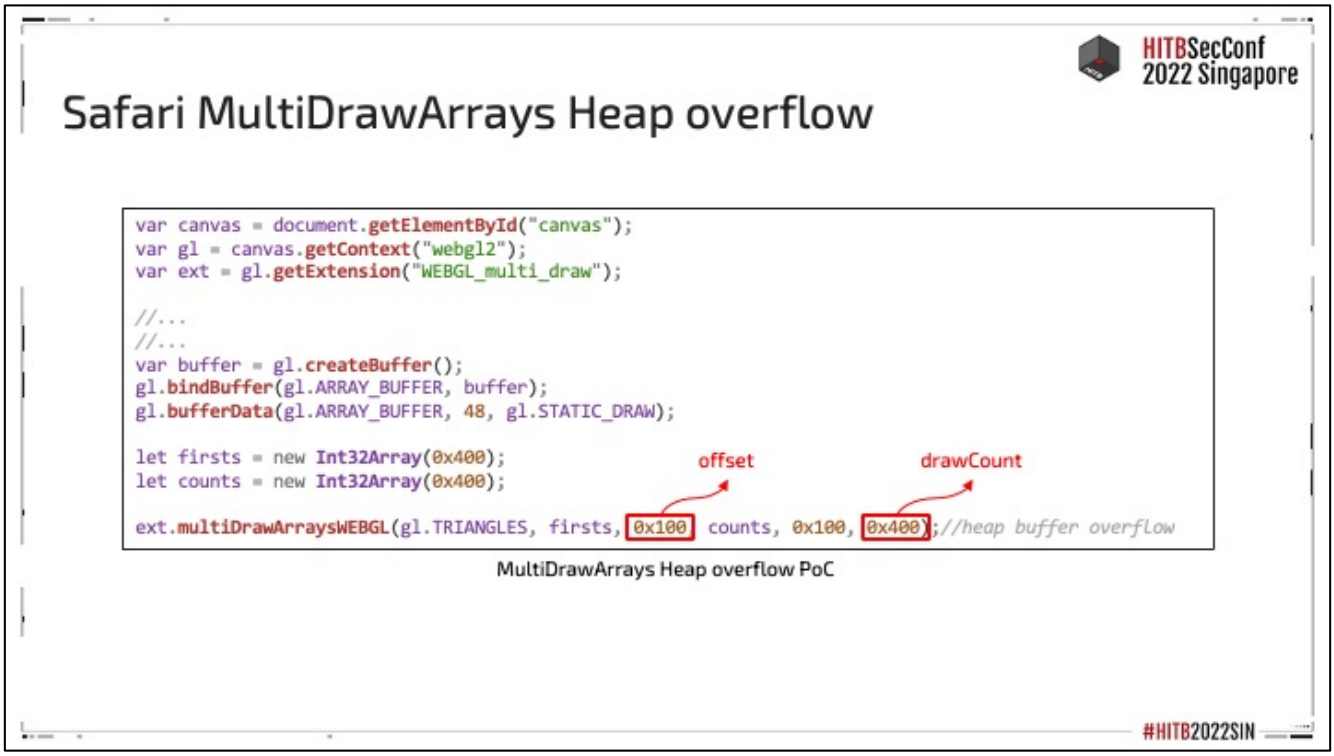


In this code, to check drawCount and Offset are less than the size

It means, the size variable should be greater than drawCount and offset.

But the problem is, if the sum(썸) of drawcount and offset is greater than size, it is not checked(첵크-드).

So, it leads to a heap buffer overflow.



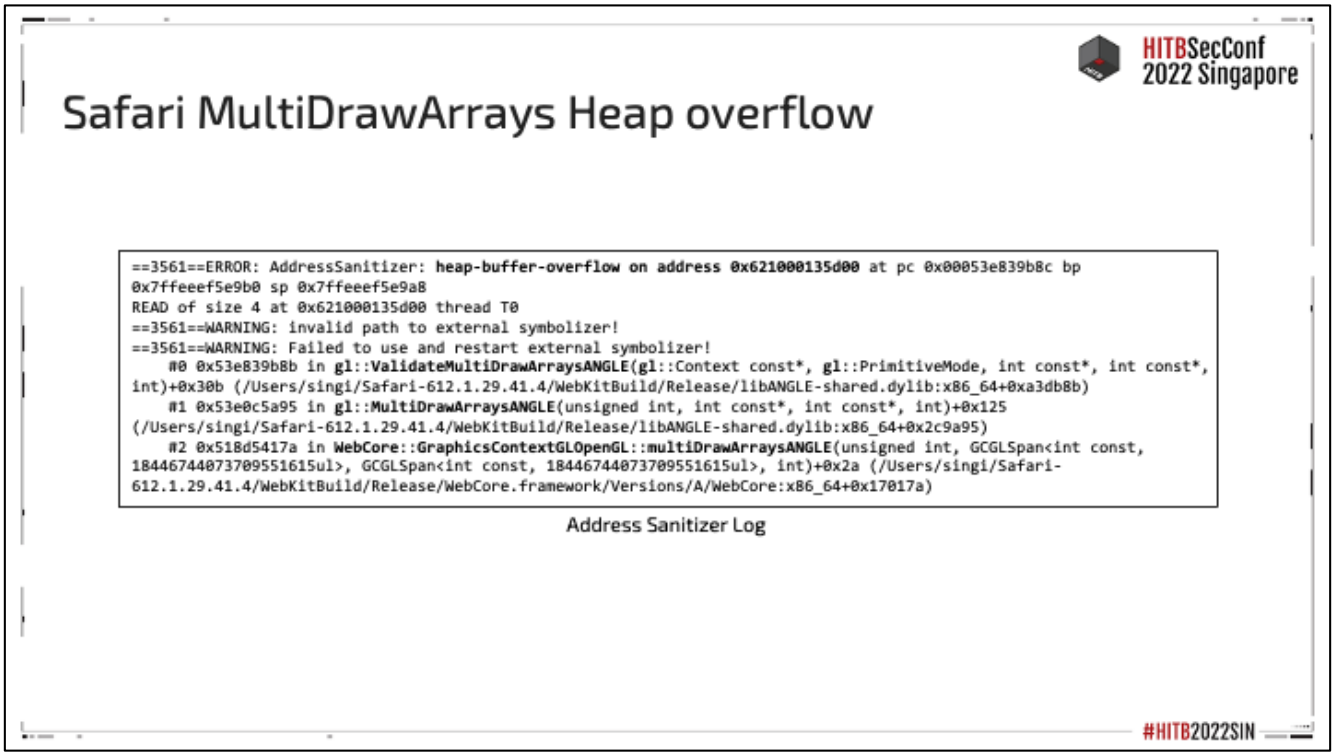
This is part of PoC code.

First, To use multiDrawArray, you must use the getExtension method like this code.

Next, call multiDrawArrayWebGL.

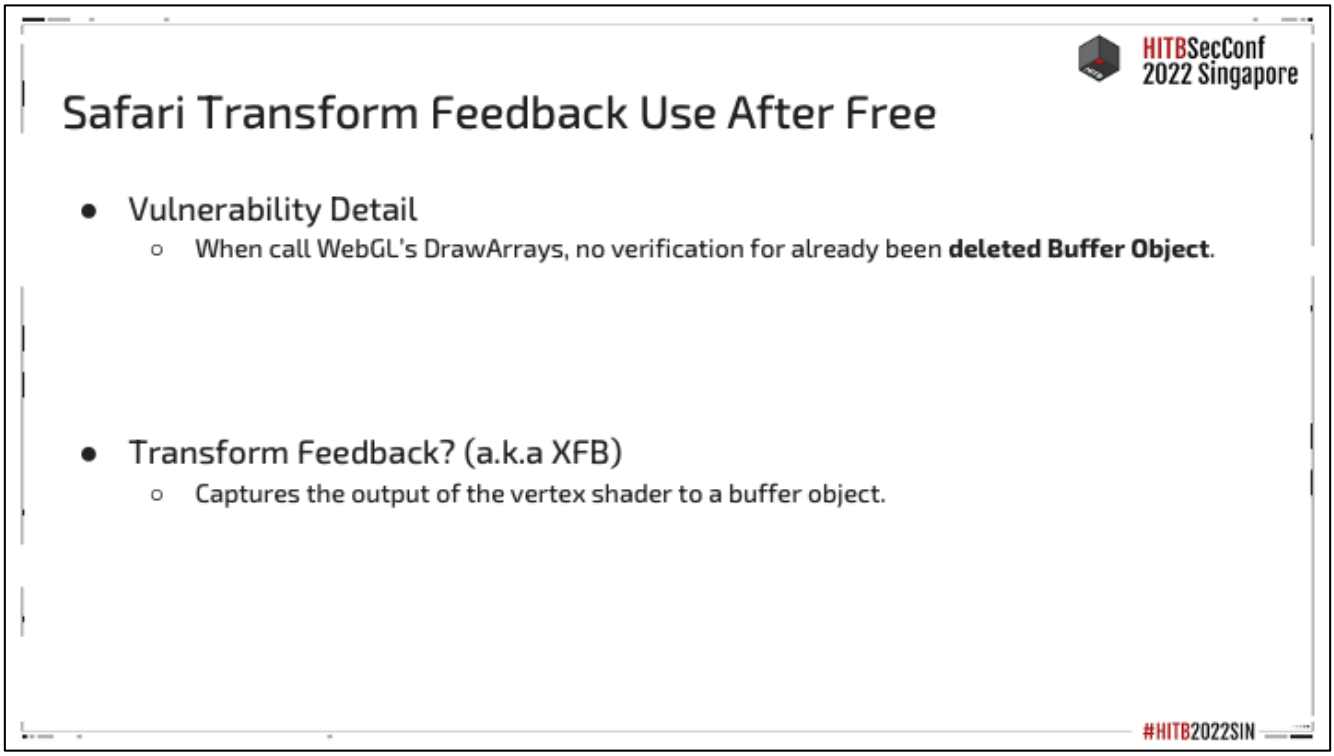
At this point, we passed drawCount and offset bigger than the provided array size.
(슬라이드에 offset / drawCount 보여주면서)

So, as a result, a heap overflow occurs(어컬스).



This is the result getting with Address Sanitizer.

And this bug can trigger the chrome browser as the same PoC.



Next, We will deep digging into the exploitable WebGL bug.

This bug also occurs when validating several states in DrawArrays in ANGLE.

This bug occurs in the transform feedback feature(피처).

Transform feedback uses a buffer object of ANGLE.

To Validate transform feedback / before calling drawArrays. / At this time, the already removed buffer is used.

Before rendering, / in drawArray method / to validate about transform feedback object.

At this point, can use the already removed buffer object in the transform feedback object.

Transform feedback uses a buffer object to store the output of the vertex shader.

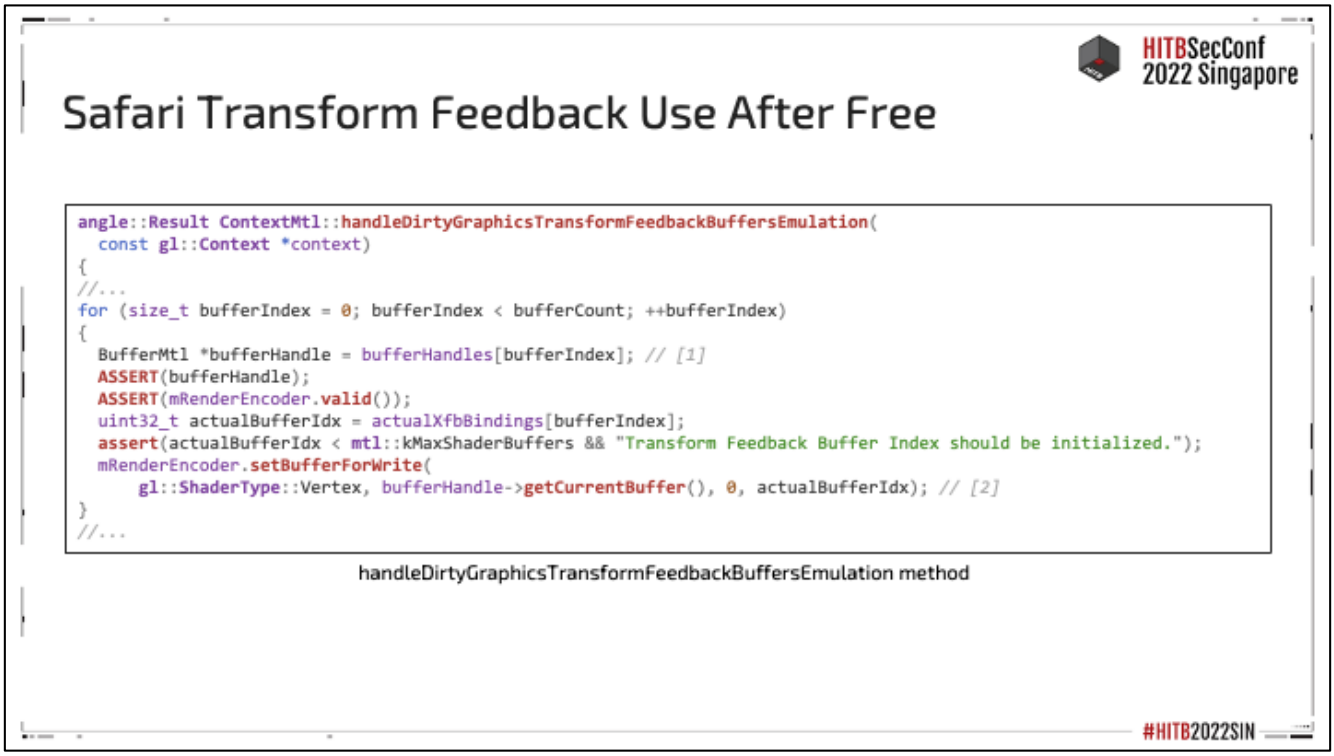
The stored value in the buffer object is used by the GPU for faster drawing.

Also, for your information,/ Transform feedback is available(어베일러블) starts

with WebGL2.

And, among the three main browsers / safari was the latest to support it. Since Safari 15(피프틴)

However, I never used WebGL for anything other than bug hunting, so I don't know the details of how it works.



The bug pattern is similar to the getSamplerTexture bug.

In Comment 1, the code finds the Buffer object from the buffer handles array.

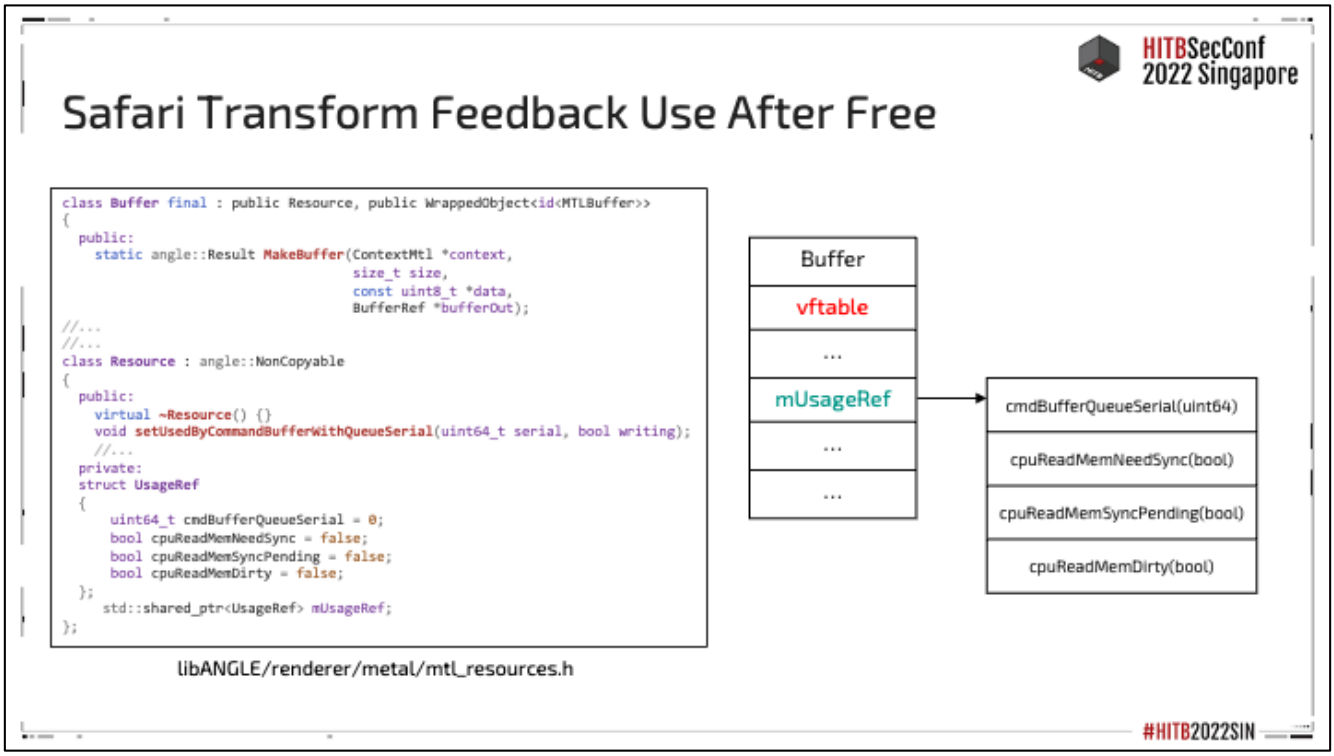
However, the buffer handle may contain the removed buffer object.

And as a result the crash occurs(어컬스) in Comment 2,
when getCurrentBuffer accesses / the already removed buffer object.

As we will learn more / about it / in the Exploitation part, The user can control the return value of getCurrentBuffer / through memory re-allocation.

After that, it is possible to exploit using the reallocated value(리얼로케이티-드 벨류) in the setBufferForWrite(셋버퍼포라잇) method.

Let's take a look at this method.



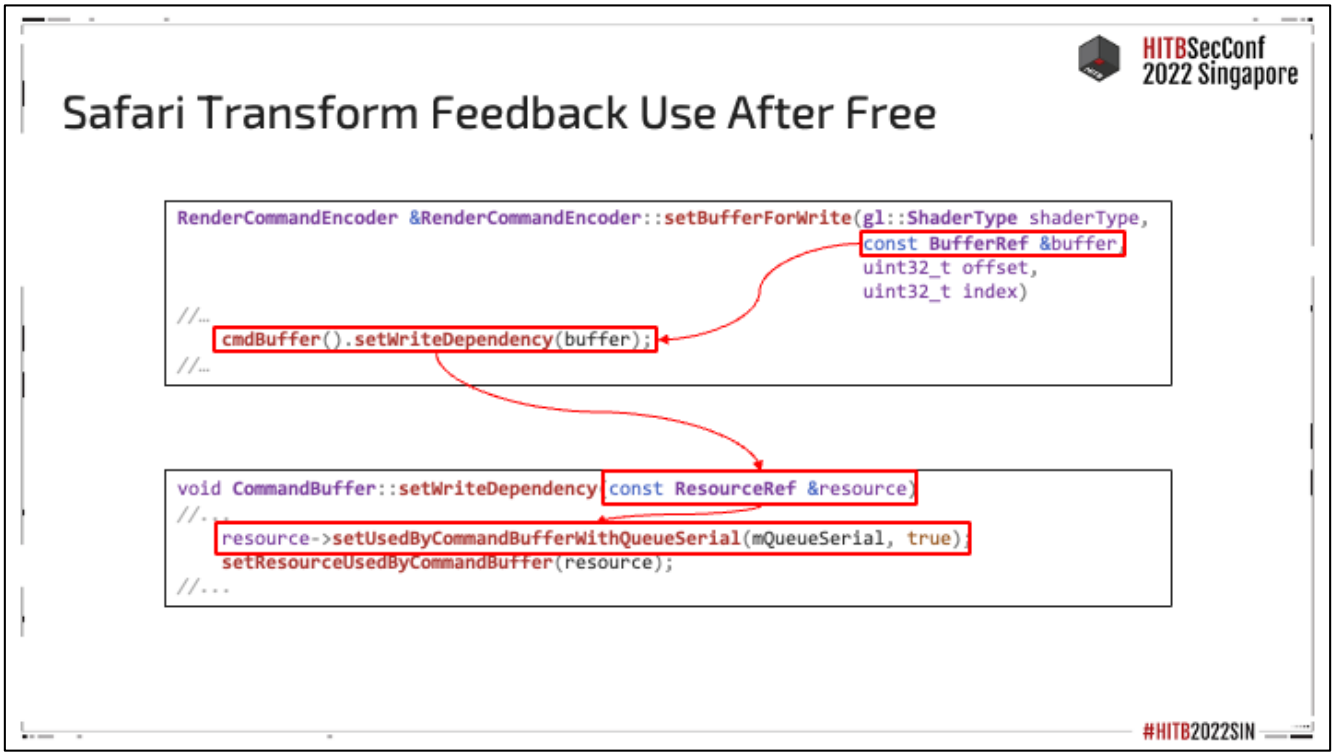
Before analyzing(애널라이징) in detail, look at the / Buffer and Resource Class.

As you can see, Buffer Class inherits(인헤릿-트) from Resource Class.

And in the Resource class, define a structure called mUsageRef.

This Structure is important for our exploit.

We will use this Use after free bug to puts the attacker's desired(디자이얼드) value into the mUsageRef field.



Now, analyze(애널라이즈) where the crash point.

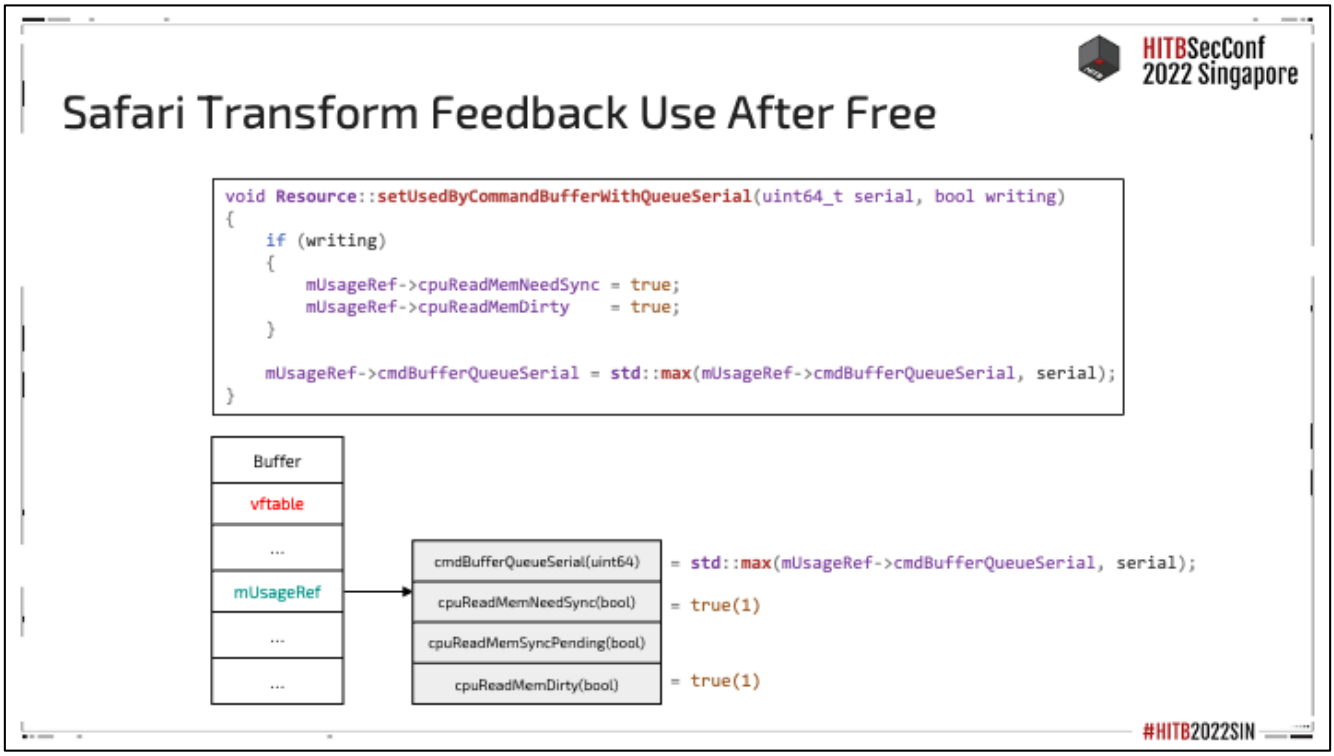
If we remember correctly, the crash occurs in the setBufferForWrite method.

If you look at the method code, / it receives the buffer value, and / calls the setWriteDependency method.

After that, the member method of the resource Class, call setUsedByCommandBufferWithQueueSerial.

At this point, the second argument is set to true.

Next, let's look inside this method.

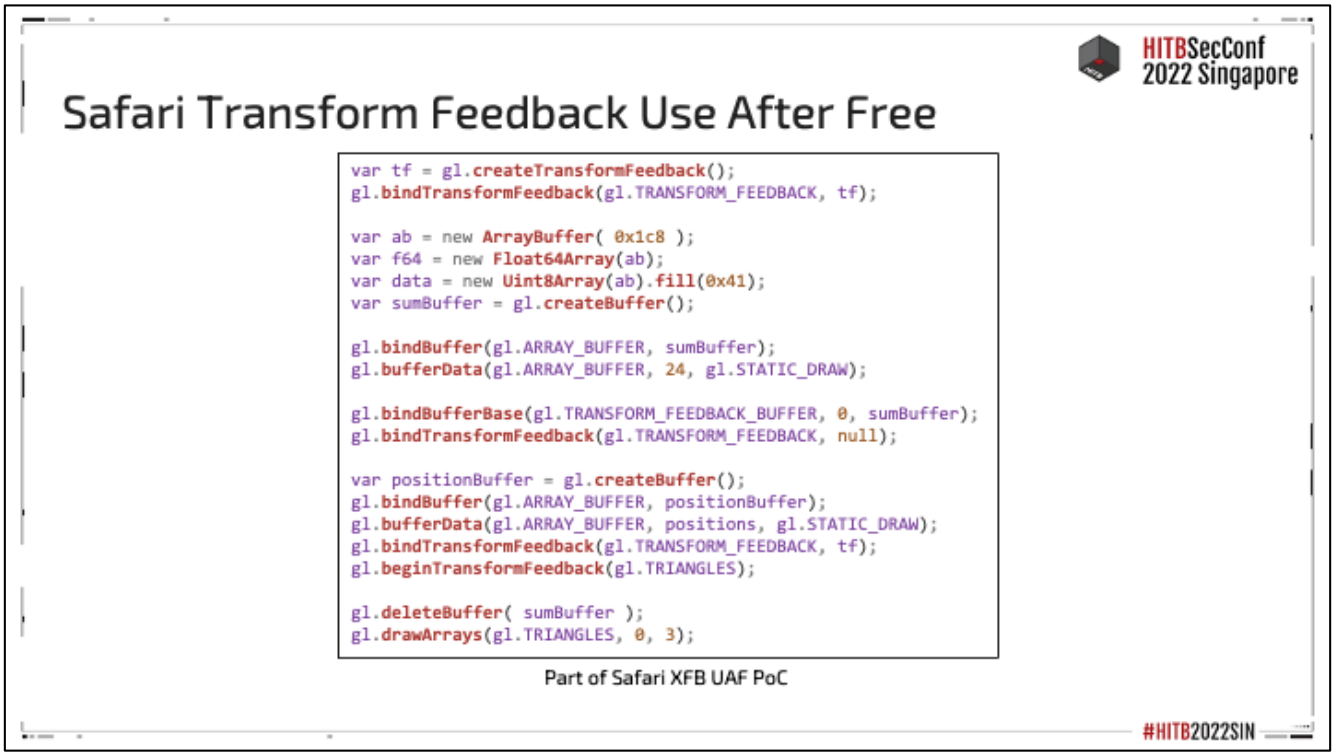


since(씬스) the writing argument is true, it enters the if statement.

So, a value is put in the member variable of the mUsageRef. As you can see, both variable types are Boolean types.

After that, put the value in the cmdBufferQueueSerial field. at this point, the return value of the max method was always 6. I Found this by debug.

Still, I'm not sure exactly what this field does. ☺

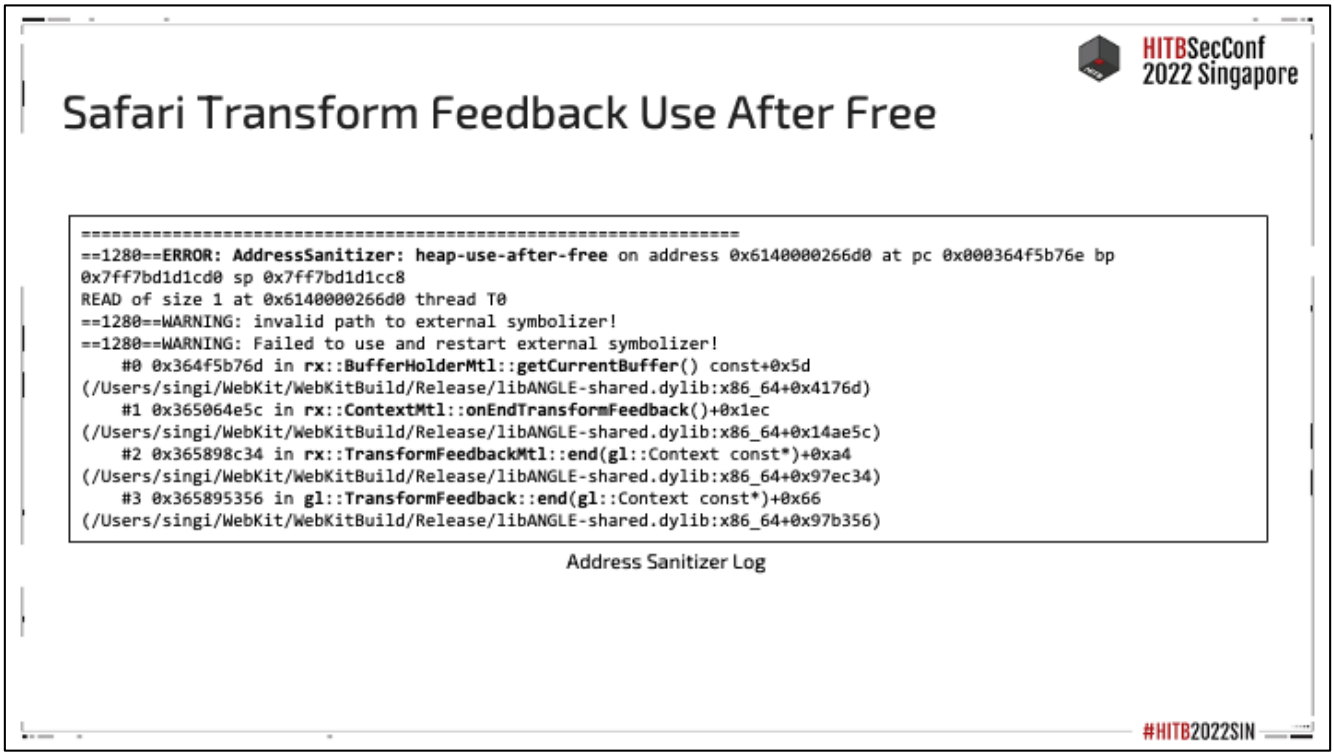


This code is part of PoC to trigger the bug.

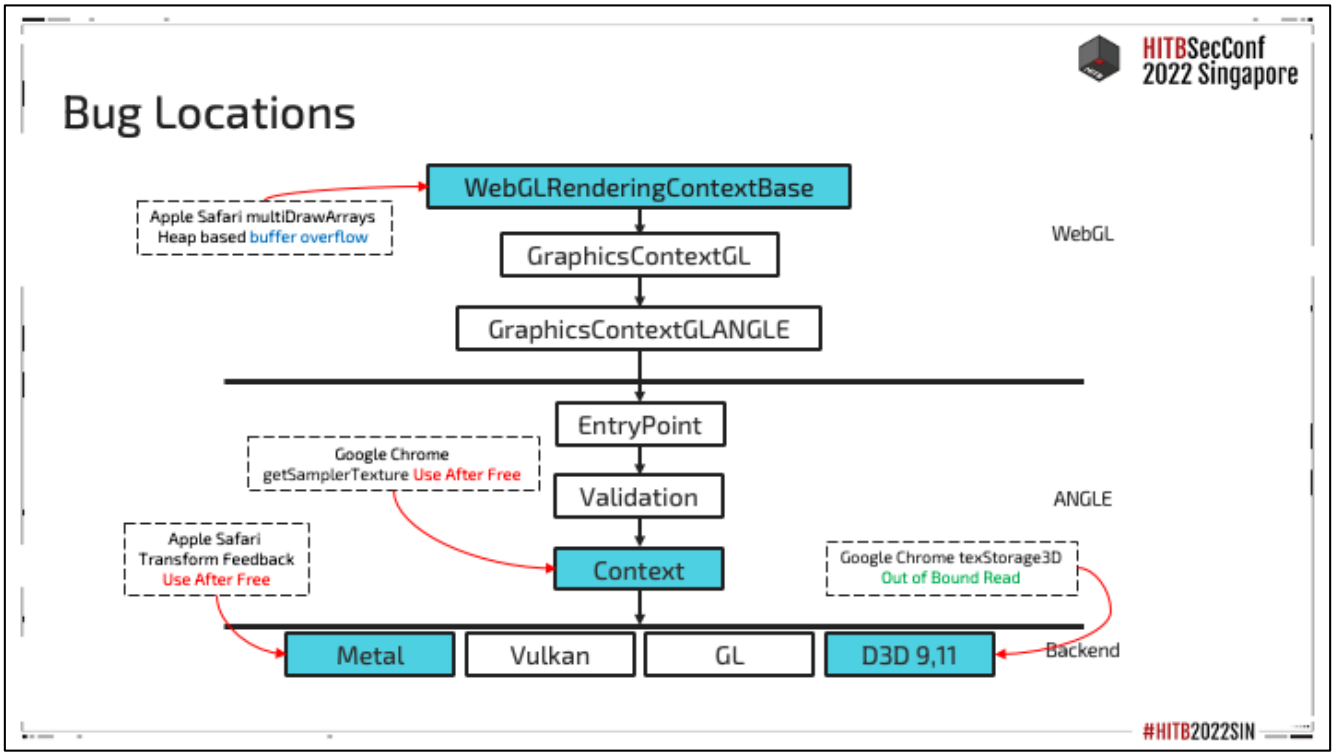
In Shortly(시옷-을리), / This is the basic code of / use transform feedback feature in WebGL2.

As you can see, / this. (drawArrays 가리키며) before calling drawArrays, just removed the Buffer object.

This removed buffer object was / the bound to transform feedback target.



This is the result with Address Sanitizer.



So far, a total of four bugs have been analyzed(애널라이즈드).

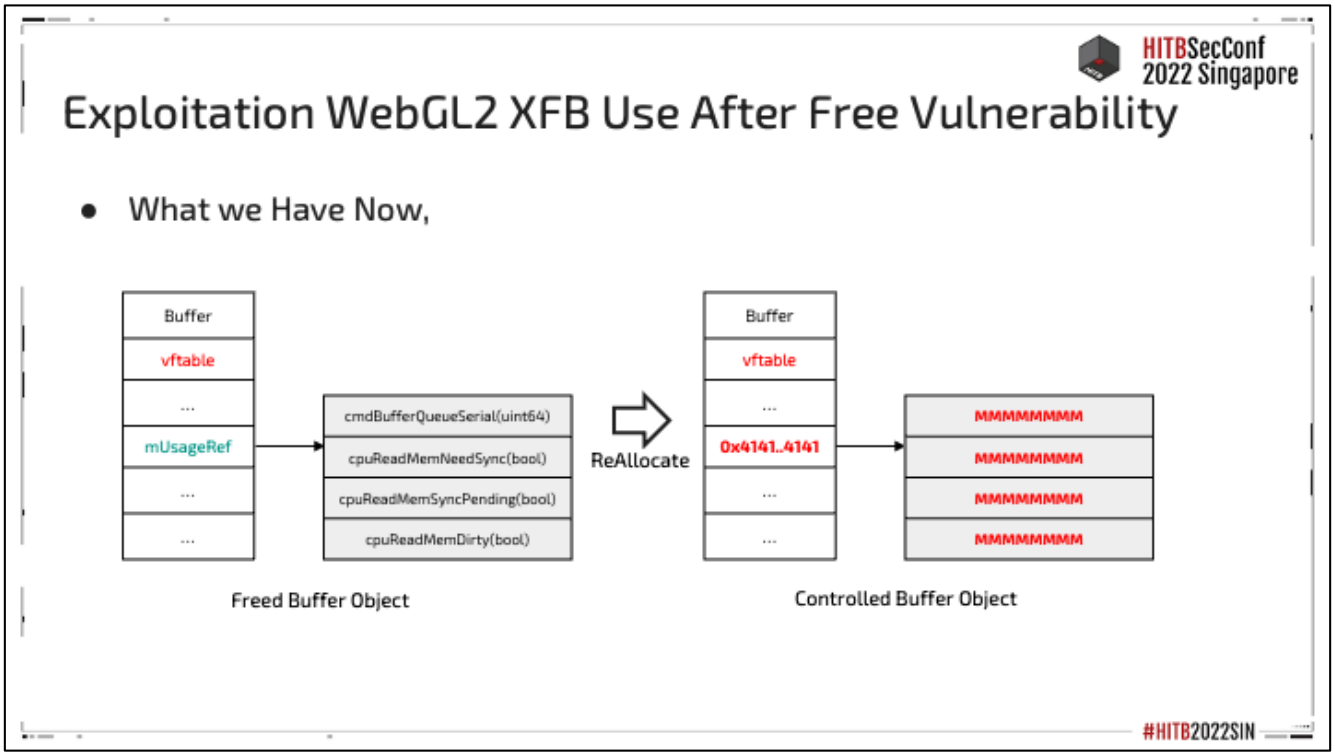
This is the location of bugs in the WebGL architecture, we looked at today

I didn't have time to cover them all, but ANGLE is still one of my good attack vectors.

because it has to communicate with the GPU / needs more permission than a renderer, and whenever the / GL standard is updated, the WebGL feature is also updated.

Also, WebGPU is / to be officially released soon.(오피셜리 릴르즈-드 순) This part will be a good attack vector.

Next, I will discover the details of how to exploit the bug we just looked at.



So, Finally we reach the more interesting part.

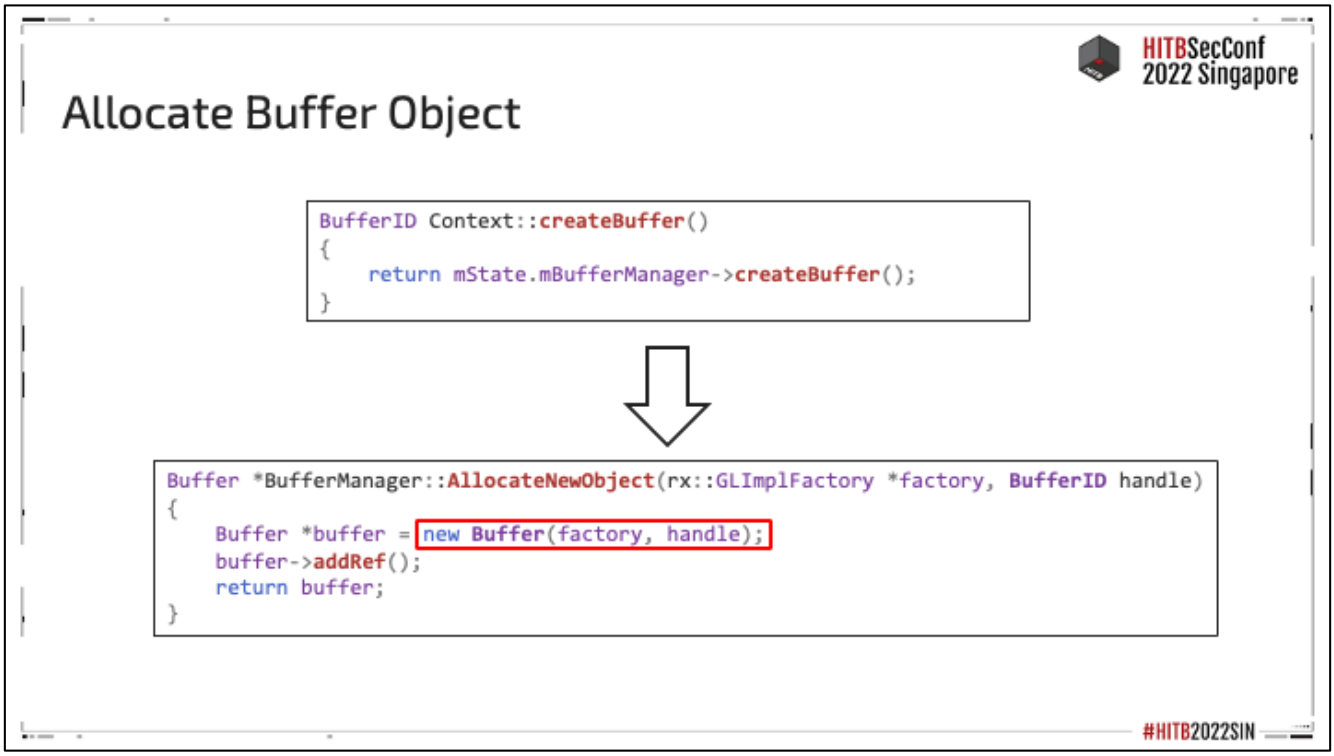
In this part, I explain how to exploit the transform feedback bug that we looked at. (지금부터 exploit에 대한 상세한 방법을 설명한다.)

First at all, let's recap to what we have. (우선, 우리가 지금 가지고 있는 취약점부터 다시 정리해보자.)

As you can see from this figure, We can control / to variables of mUsageRef structure / of already deleted Buffer Object. (위 그림처럼, Free된 Buffer 객체의 mUsageRef 멤버 변수의 값들을 제어할 수 있다.)

Now, let's see how we can stable reallocate memory that(리얼로케잇 메모리 뎃) / has been deleted.

To find a way to reallocate(리얼로케잇), we must know how the buffer object is allocated.(얼로케이티-드)



As you can see, some Buffer object create by the createBuffer Method.

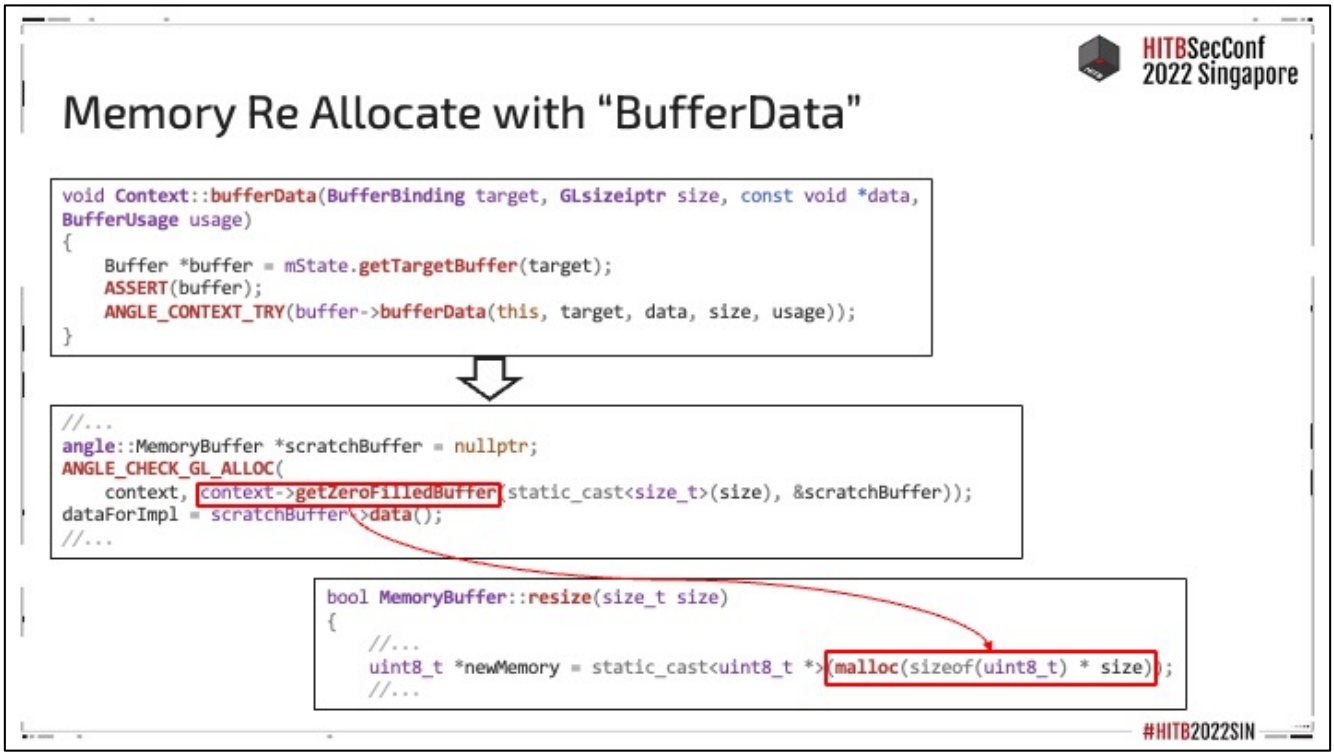
When called create buffer method, internally called create buffer of the Context class. (가리키며)

This method is also called the AllocateNewObject method / at this point, allocates(얼로케이츠) a buffer object / using a new Operator.

Also, BufferManager class does not override the new operator.

So, It is allocated(잇 이즈 얼로케이티드) / to the normal heap.(투 더 노멀 힙.)

Next, let's review to method used for reallocation.

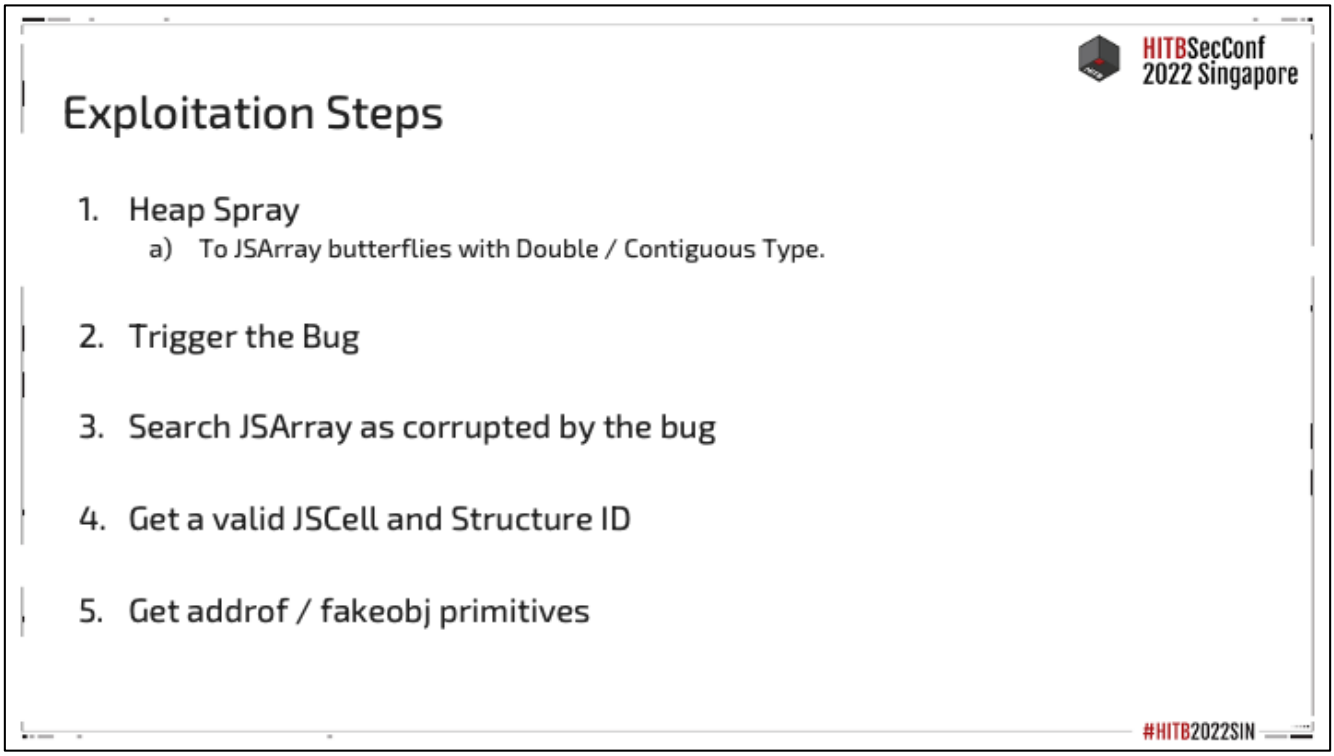


For reallocation of the free buffer object, I used the bufferData method. (Free된 buffer에 대한 재할당은 bufferData 메소드를 활용하였다.)

When we call this method / as you can see, allocate and / initialize the memory(이니셜라이즈 메모리) with getZeroFilledBuffer (겟지로필드-버퍼)

In that method, used malloc function.

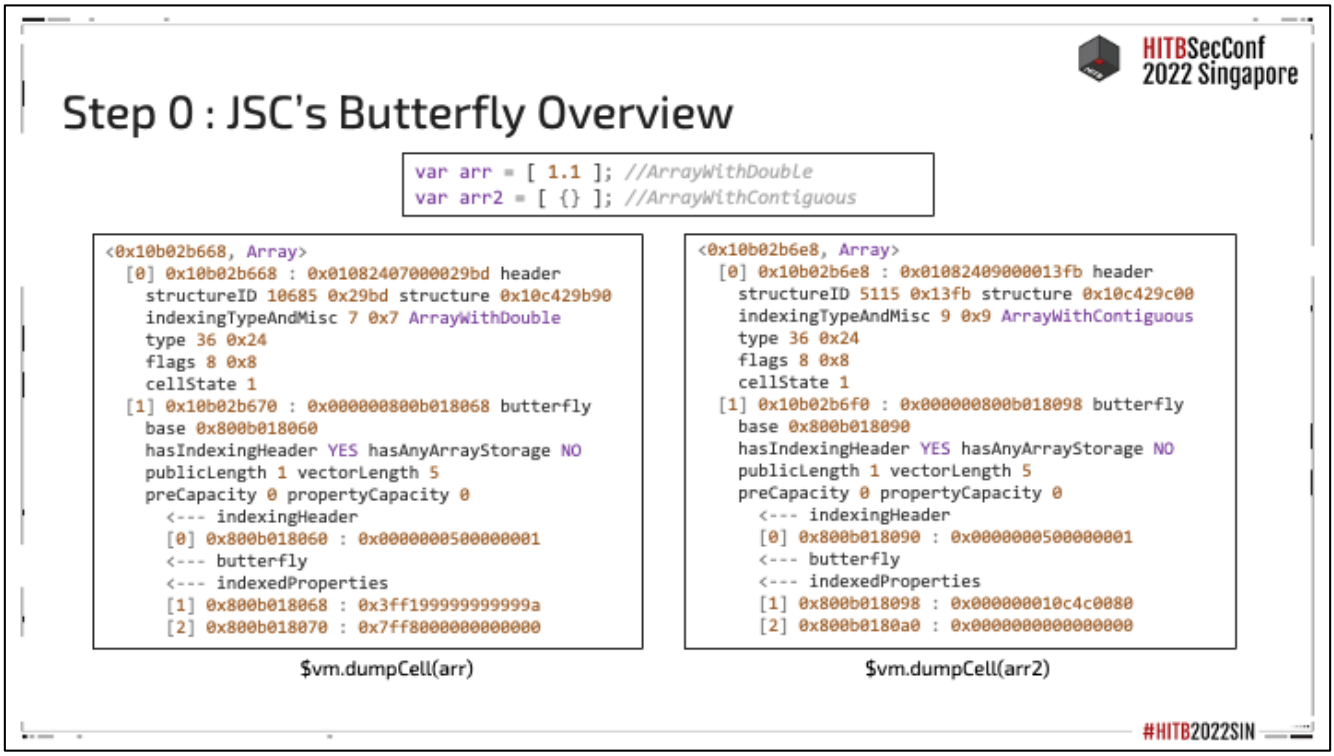
So, for the reason,(포더 리즌) We can easily allocate in the delete buffer object.



Now, look at the exploitation steps.

I explain five-step.

But, before that, let’s learn the basics of JSObject. (제이에스오브젝)



Before, heap spray step, explain to the butterfly of JSObject used in JavaScriptCore.

Check it with a simple JavaScript program.

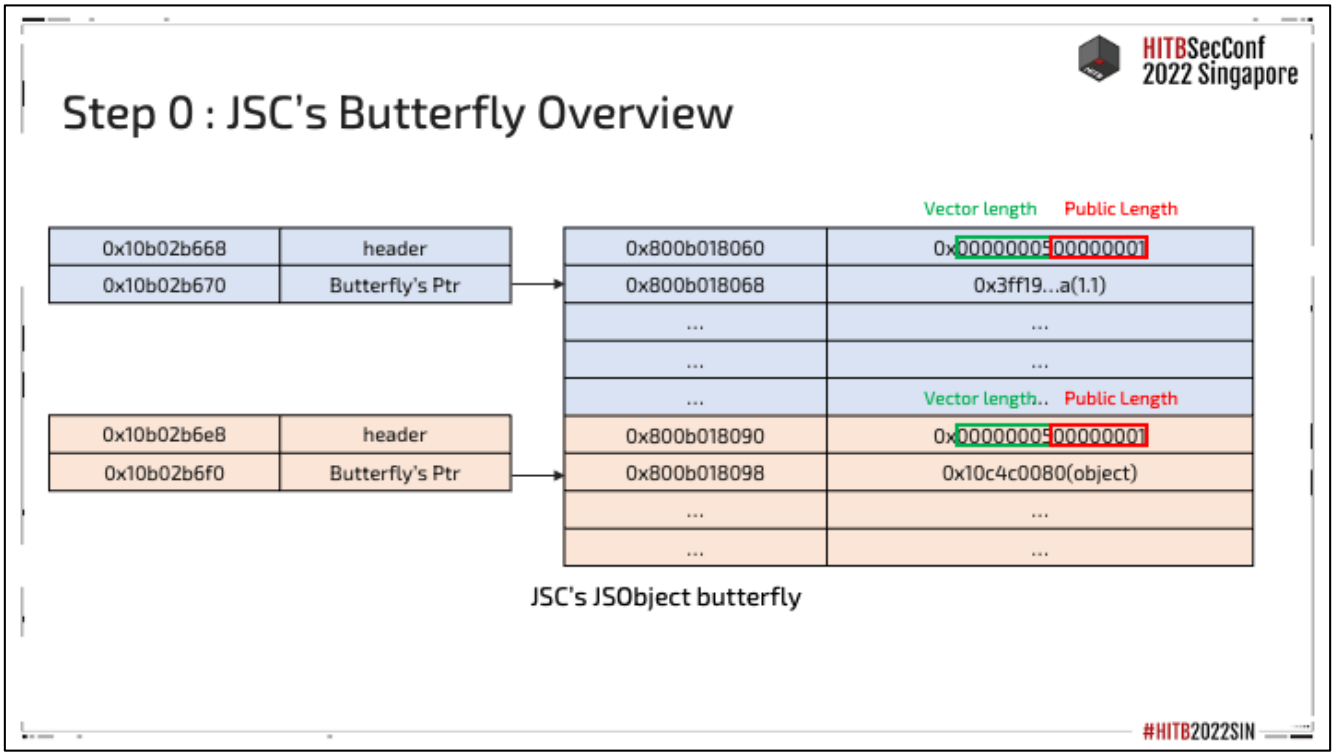
As you can see, This code(JS 코드 가리키며) simply / define two arrays and / puts values of different types..

After that, check the results of arr(어레이) and arr2(어레이투) with the dumpCell function. (dumpCell function is JavascriptCore Native Helper function)

So now, let's look at the result of the array. Array Type became ArrayWithDouble(가리키며) because it puts a value of Double Type.

Next, Array2 Type became ArrayWithContiguous(컨티규어스) (가리키며) because it puts a value of Object type.

Now let's see what this code looks like in memory.



This figure is the memory representing the two arrays create in the previous.

The Blue color is a double type array, and the other is a contiguous type array.

First, looking at the double type butterfly, it points to a value of 1.1(원 포인 원). / And above this, you can see the value indicate to the length of this array.

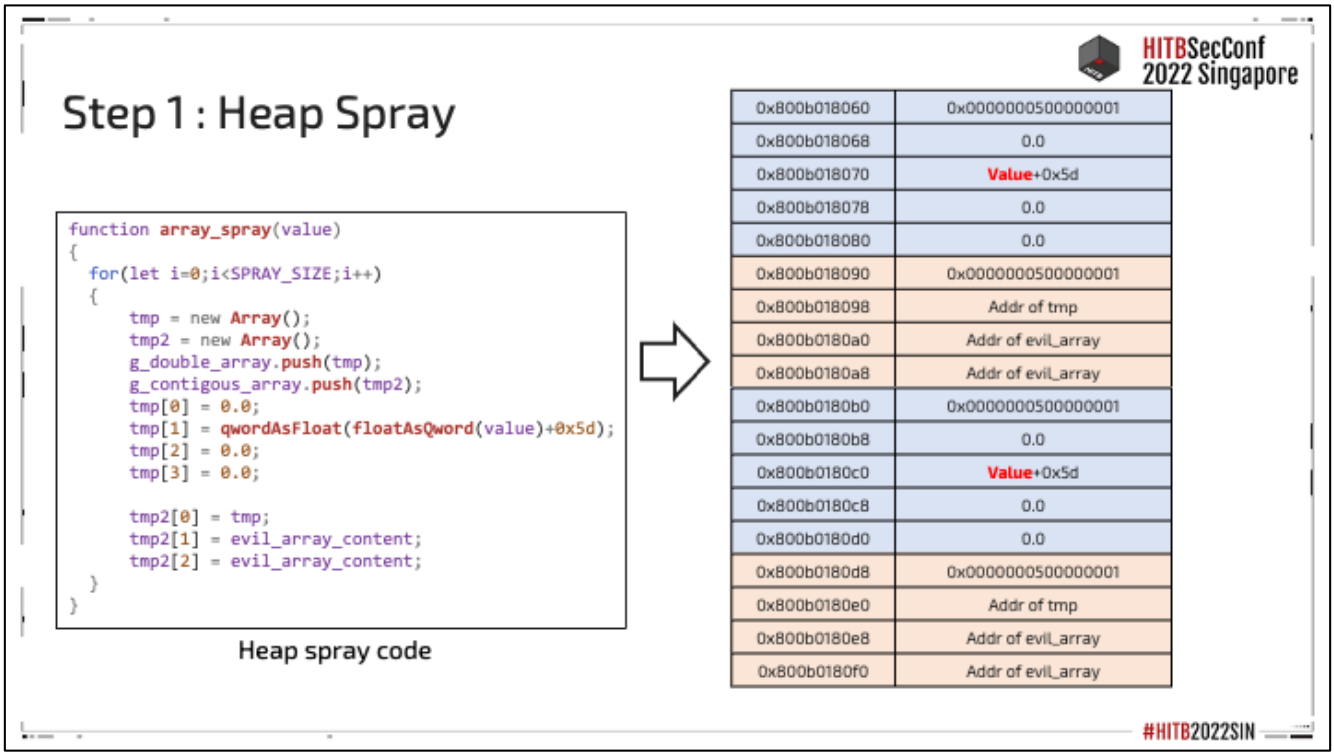
JSObject has 2 types of Length. It is called vector length and public length.

The vector length is a writable size, public length is a readable size. And,,,,, when we used the length property of Some Array, get a public length value.

In shortly(시옷-을리), When we do Out of Bound Read and Write / we need to modify the vector length.

Next, looking at the contiguous type butterfly(가리키며), it points to a value of some Object. And above this, you can see the value indicate to the length of this array.

You can see that it has the / same structure as a double type array.



Now, let's start the step1.

This code is the heap spray part in the exploit. The purpose(펄페스) of this is to spray the butterfly in memory as checked(첵크드) in Step0.

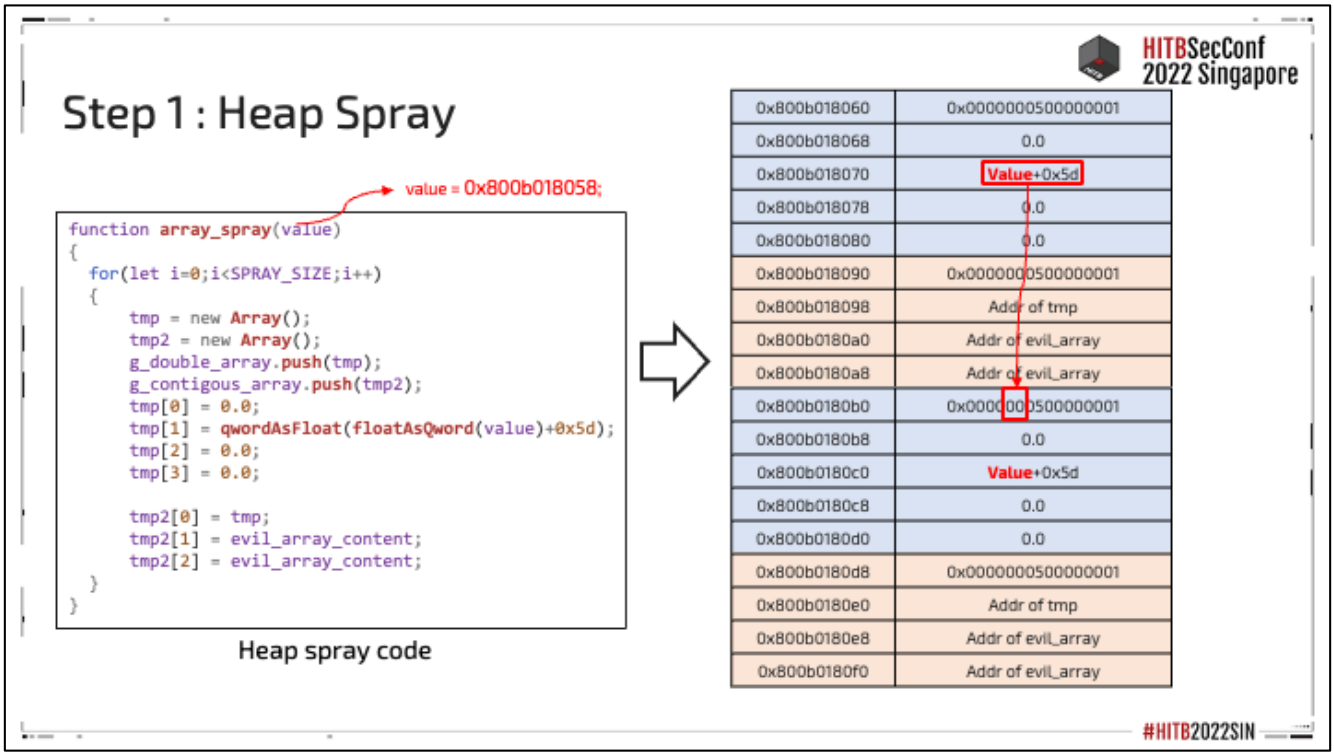
At this point, the upper few(피유) bits of the butterfly are fixed.

Using this, We can predict(프리딕-트) to sprayed butterfly address.

When step1 is executed(엑스큐티드) / many butterflies are created in memory.

As you can see, the sprayed memory has the same shape / as the / one on the right.(원 온더 라이츠)

keep this shape / in the mind. It uses until the end of this talk.



Now talking about spray code.

The value parameter(패러미터) is important. (가리키며)

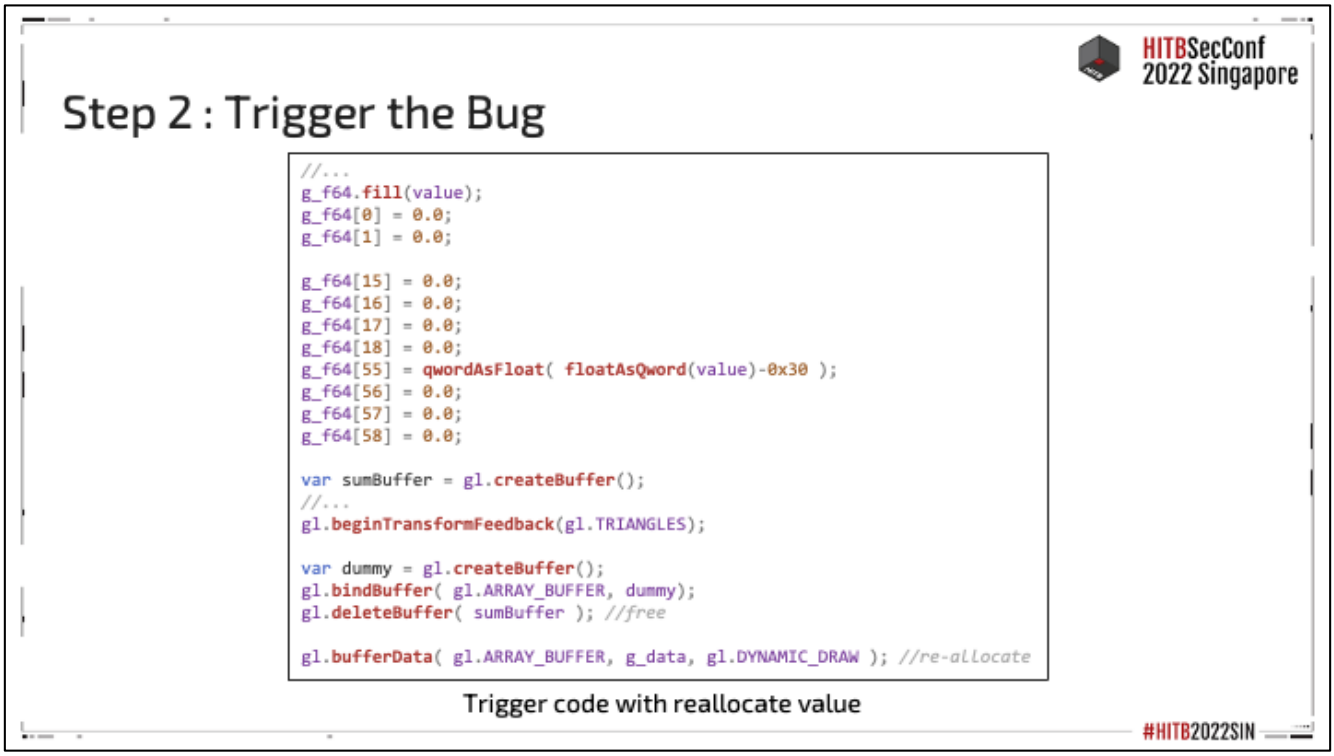
We should put the predicted(프리딕!티드) butterfly address in the value parameter.

The value must point to this memory. (value+0x5d를 가리키며)

Looking at this line(다시 +0x5d 더하는 코드 가리키며), we add offset 0x5d to the value.

If offset 0x5d is added(에디드 투 벨류) to Value, it points to the vector length of another double type array.

If the bug is triggered / after this, the value becomes the mUsageRef field of the freed buffer object.



Next, This code is part of the trigger function.

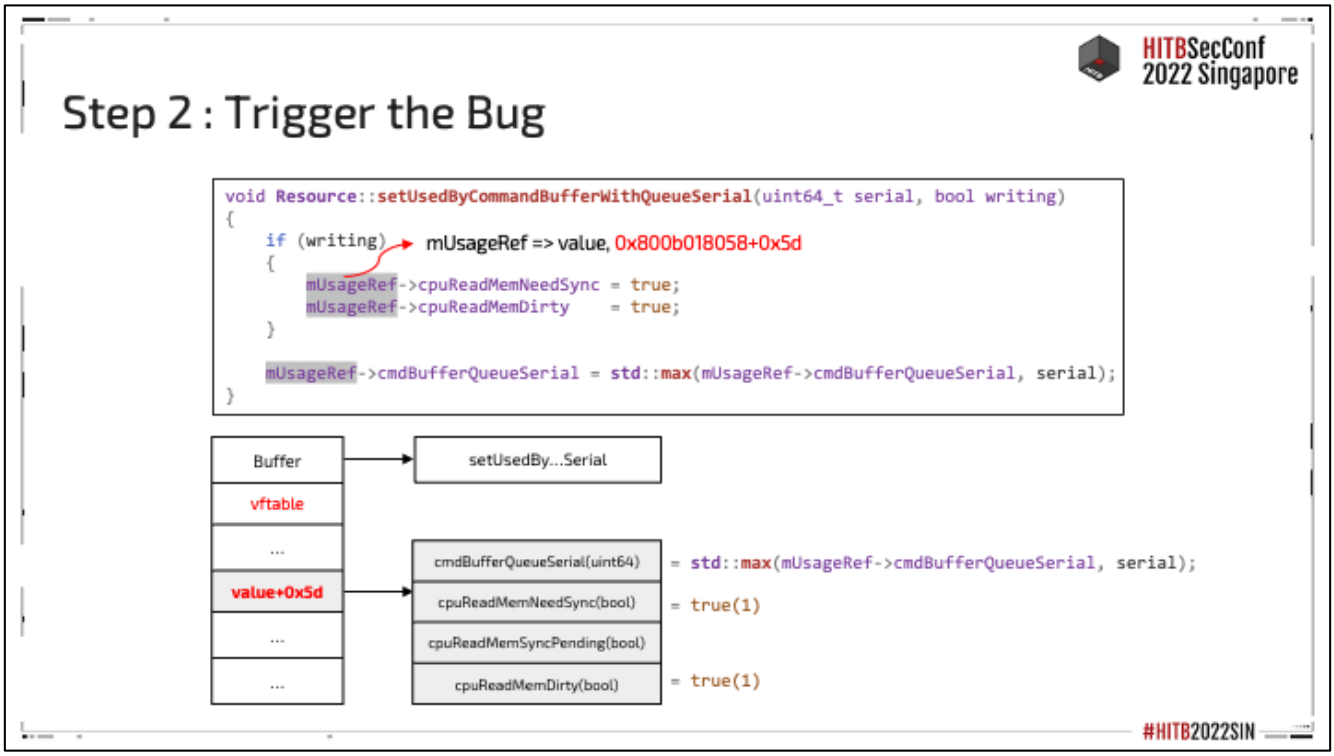
After calling deleteBuffer,(가리키며) It Attempt to / reallocate by bufferData on the / already freed memory location.
deleteBuffer 호출 후, 곧 바로 bufferData 메소드를 이용해 free된 Buffer 객체 영역에 재할당을 시도한다.

The reallocated(리얼로케이티드) value is in g_f64(가리키며 this) array.

Before trigger the bug, puts the value at index 55th(피프티-파입) of the this(g_f64 가리키며) array.

After that, we trigger the bug.

Index 55th is the location of the mUsageRef field. I found by debug.

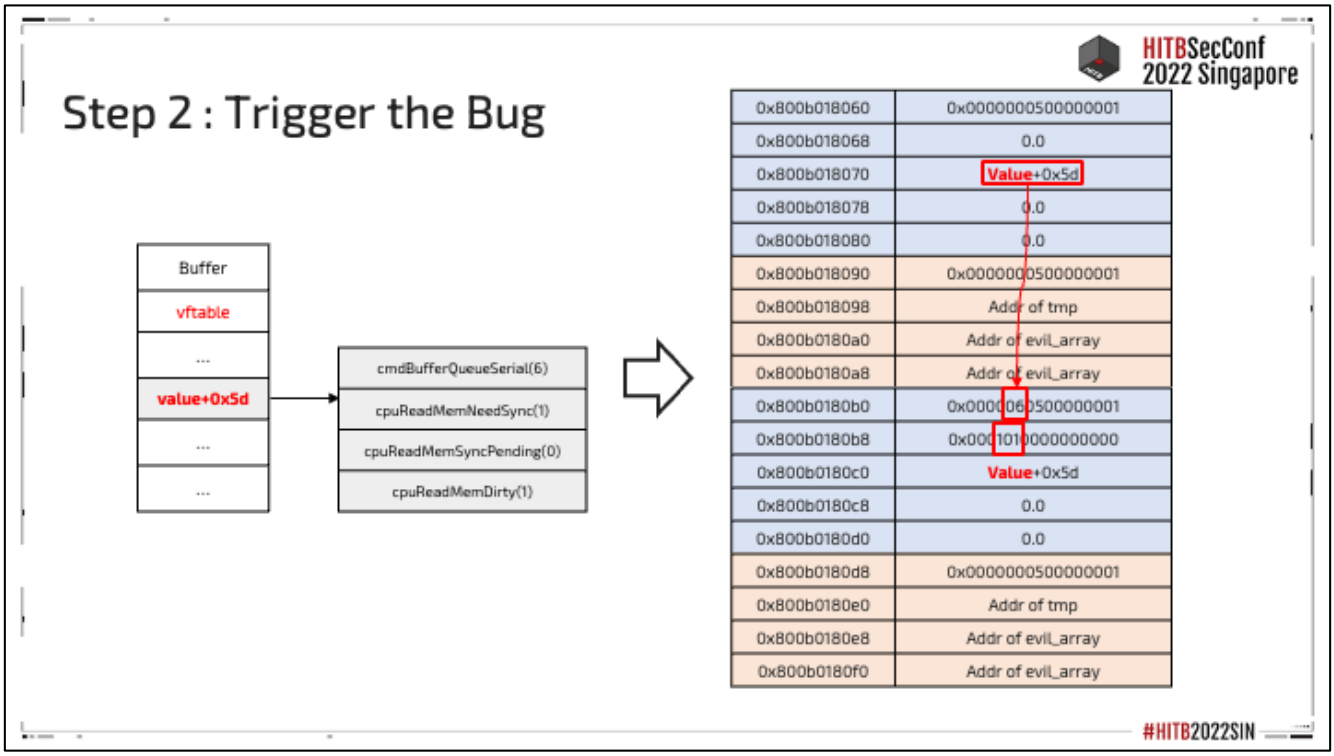


As we know, When trigger the bug, we can reach this method.

At this point, mUsageRef should be replaced with exactly(이그젝-클리) the value we want.

the target we want to change is the / vector length and index 0 of the double type array in the sprayed butterflies.

Let's look at this a / little more. (렛츠 룩앳디스 리들 모어)
이것을 조금 더 자세히 알아보겠습니다.



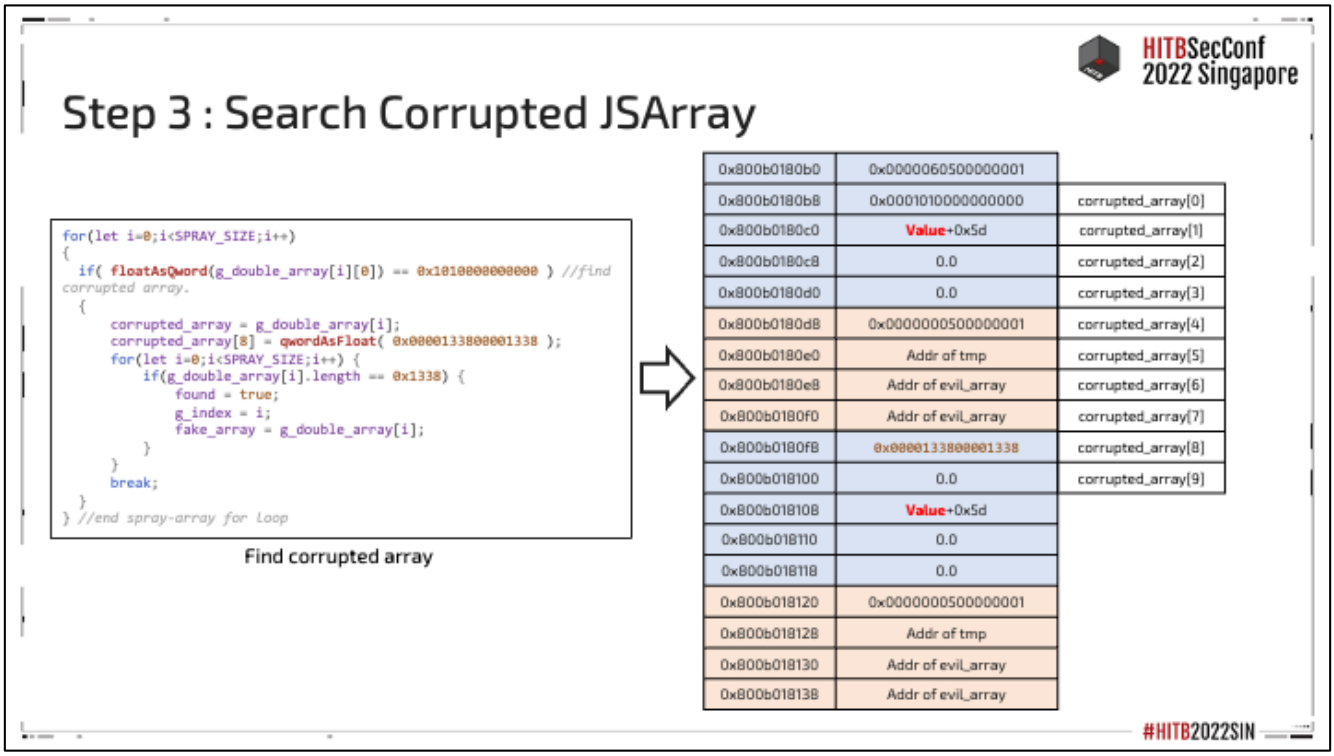
This figure shows what the Trigger function does.
이 그림은 (가리키며) Trigger 함수가 해야 하는 일을 나타냈다.

If the mUsageRef field has been reallocated(리얼로케이디드) / it update cmdBufferQueueSerial, cpuReadMemNeedSync and cpuReadMemDirty fields. (필드들 가리키며)
mUsageRef 필드 값이 재할당되었다면, 이런 필드의 값이 업데이트 된다.

As I mentioned, the vector length of double array and the index 0 are changed to these values. (가리키며)
아까 말했듯이, double type 배열의 vector length와 그 배열의 0번째 인덱스를 이런 값들로 변경한다.

So, The vector length was changed from 5 to 0x605(씩스티-파이브). Now, this array can be Out of bound read and writes.
vector length가 5에서 0x605로 변경되었다. 이제, 이 array는 out of bound read와 write가 가능한 상태다.

Using this, to proceed(프로-씨-) with next steps.
이것을 이용해 나머지 step을 진행한다.



For now, we can call this array that has been resized by a bug a “corrupted array” (가리키며)

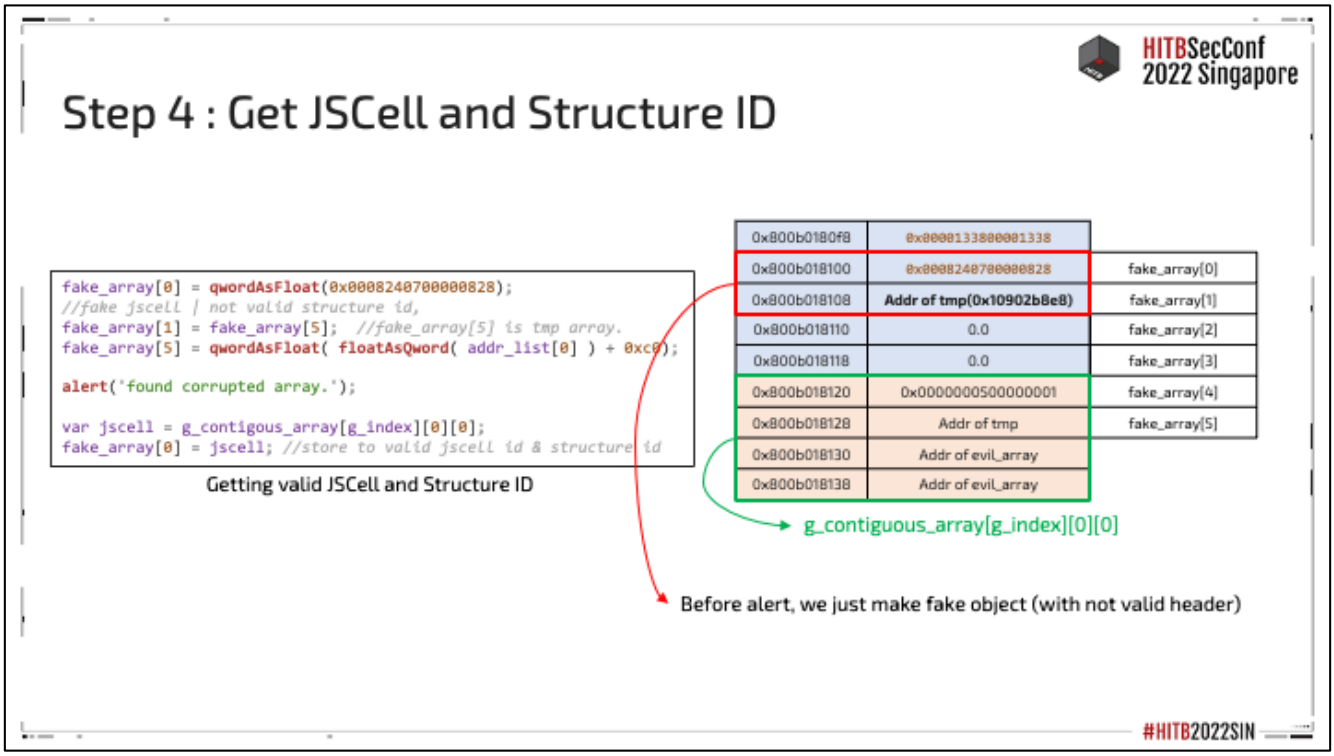
So, look at this line, (PoC 중 일부 가리키며) the index 8 of the corrupted array has the length field of another double type array.

We change the length of this array to 0x1338. (썰틴-써리에잇)

So now we have a fake array of the large size.

Now we can do something with this fake array.
이제, 이 fake array로 재밌는 일을 할 수 있습니다.

One of them / for stable exploitation / we can get a valid StructureID of JSObject.



this code is getting a valid StructureID.

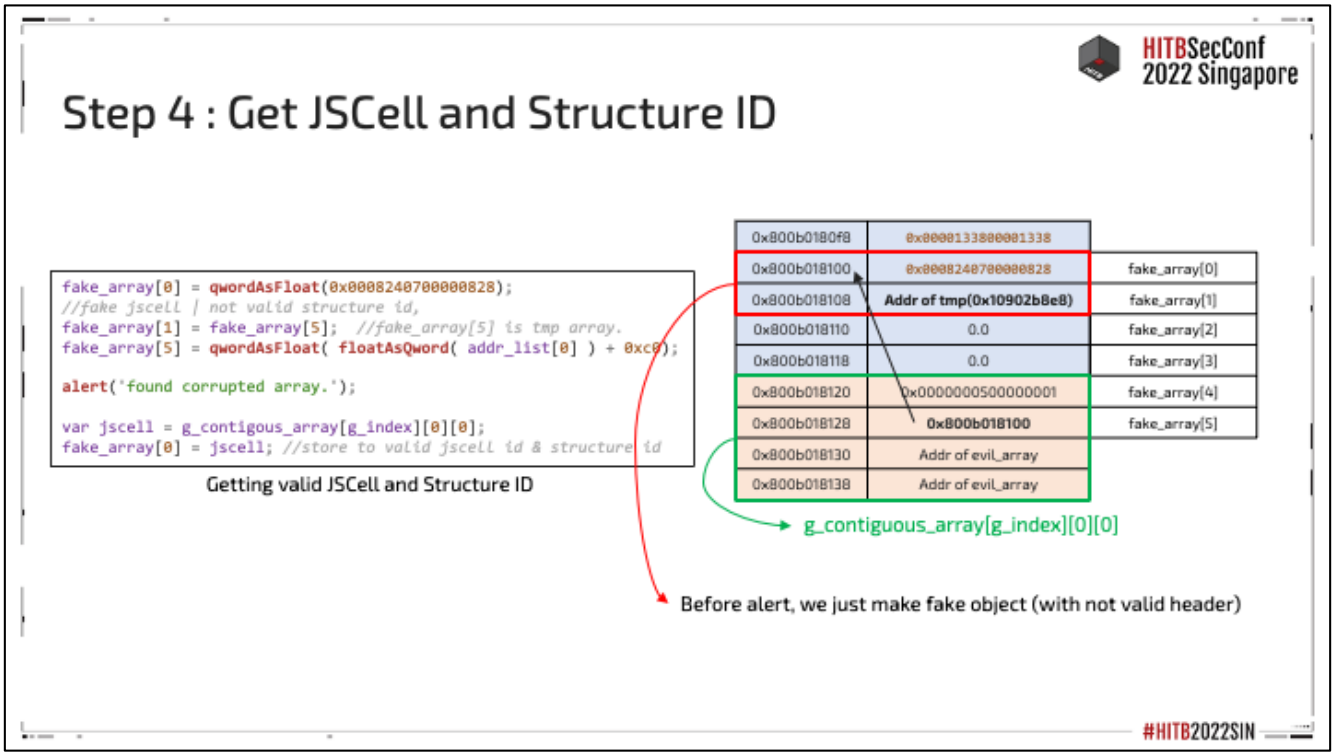
It's a bit complicated(컴플리케이티-드), but it's fun.

It is easy to / understand the code by following it,, line by line. Also, the red box is a double array, and the green box is a contiguous array.
코드를 따라가 한줄 한줄 따라가면서 보는게 이해하기 가장 편하다. 그리고 , 붉은색 박스는 double type array 이고, 녹색 박스는 contiguous array입니다.

Looking at the first line(가리키며), we put the fake strucutreID into the index 0 of fake_array. Then the memory changes like this. (우측에 fake id 넣어준 곳 갈키면서)

In the next line, we put the index 5th value of fake_array into the index 1(one) of fake_array.

After that, change the value of index 5th of fake_array to value plus 0xc0.
그 후, 3번째 줄에서 fake_array의 5번째 인덱스의 값을 value+0xc0으로 변경합니다.



This is the memory shape that / was executed(엑스큐티드) before the alert. In browser exploitation, alerts is also good breakpoints. (of course, if you have done your exploit, should remove the alert.)
이게 alert 전까지 실행 된 메모리 모양이다. 사실 alert은 아주 좋은 breakpoint 이기도 하다.

So, As you can see, the value of the index 5th of fake_array points to the index 0 of fake_array.
볼수 있듯이, fake_array의 5번째 인덱스의 값이 fake_array의 0번째 인덱스를 가리키고 있다.

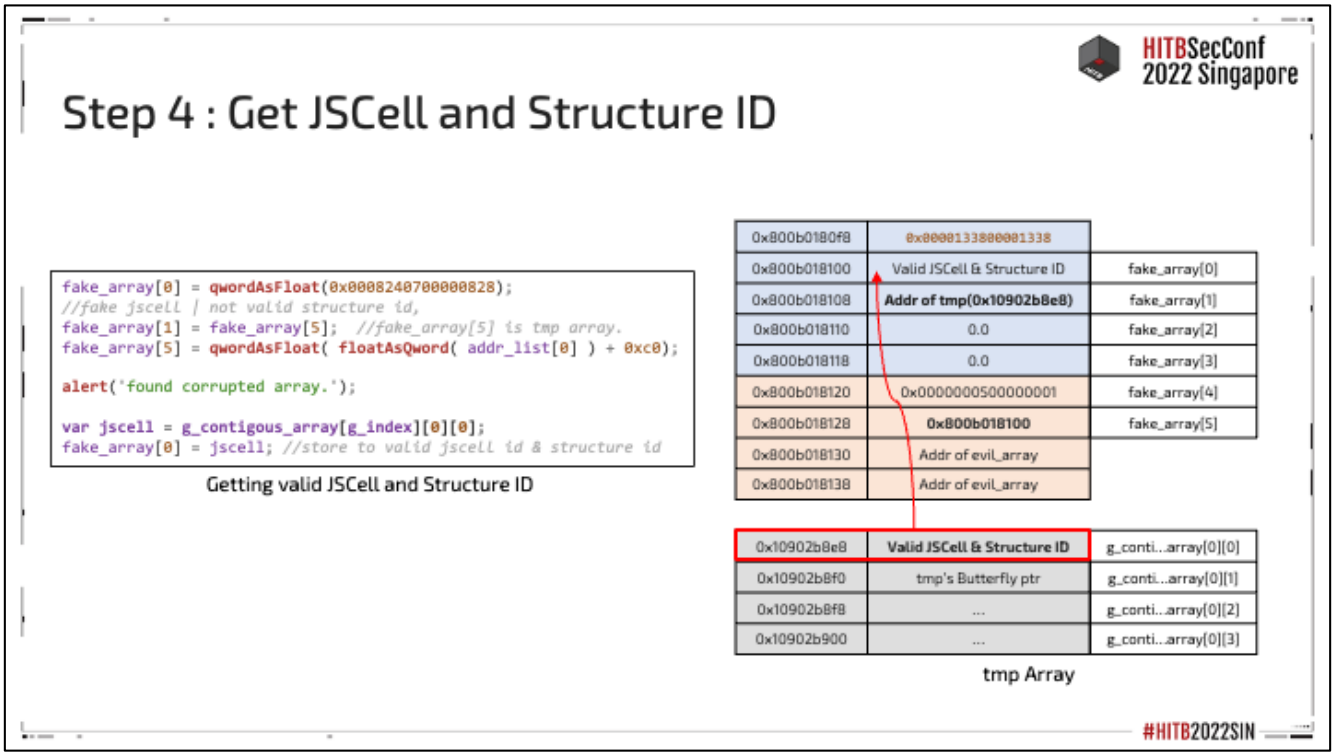
As I mentioned, the green box is g_contiguous_array, we are currently replacing the index 0th of this contiguous array.
녹색 부분은 g_contiguous_array로, 우리는 현재 이 contiguous 배열의 0번째 객체를 바꾼 것이다.

So, when we access / the index 0 in this contiguous array, it will recognize(레킥나-이즈) the red box as an object.
그래서, 우리가 이 배열에 0번째 인덱스에 접근하면, 붉은 박스를 객체로 인식할 것이다.

Recognize this address(0x…100 가리키며) as the StructureID of the fake object, then, the address of the temp array is recognized as a butterfly pointer.

이 주소를 가짜 객체의 StructureID로 인식하고, 그 다음 실제 array의 주소를 butterfly로 인식한다.

For this, what will happen / if we read the index 0th of the contiguous array? The answer is in the next slide.

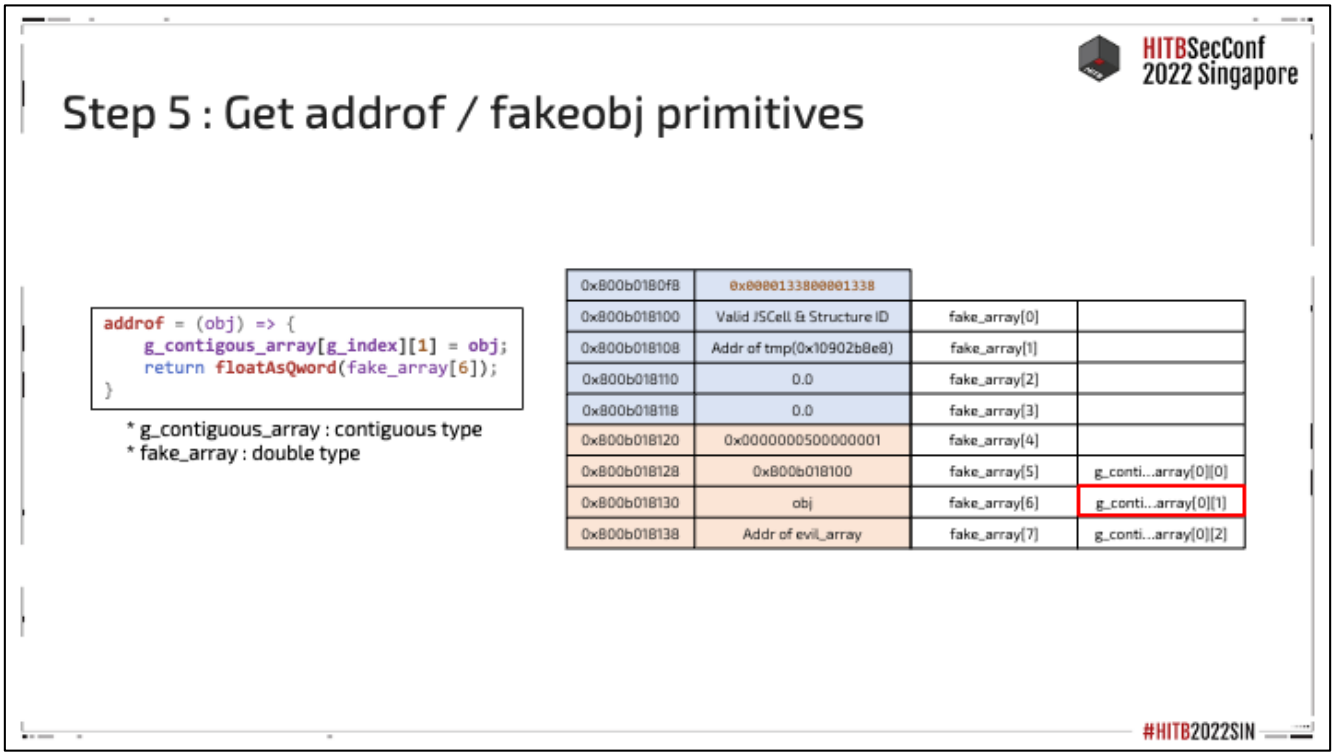


Looking at the next line, Read the index 0 value of g_contiguous_array.
다음 라인을 보면, g_contiguous_array의 0번째 인덱스 값을 읽는다.

At this time, the index 0 value is fetched based on the changed butterfly.
이 때, 변경된 butterfly를 기준으로 0번째 인덱스 값을 가져온다.

For now, as you can see, butterfly points to the tmp array address.
변경된 butterfly는 tmp array 주소이다.

So, get structureID stored in the actual tmp array. Like this… So You can use this.
따라서, 실제 tmp array에 저장된 valid structure ID를 가져온다. 이것을 이용하면 된다.

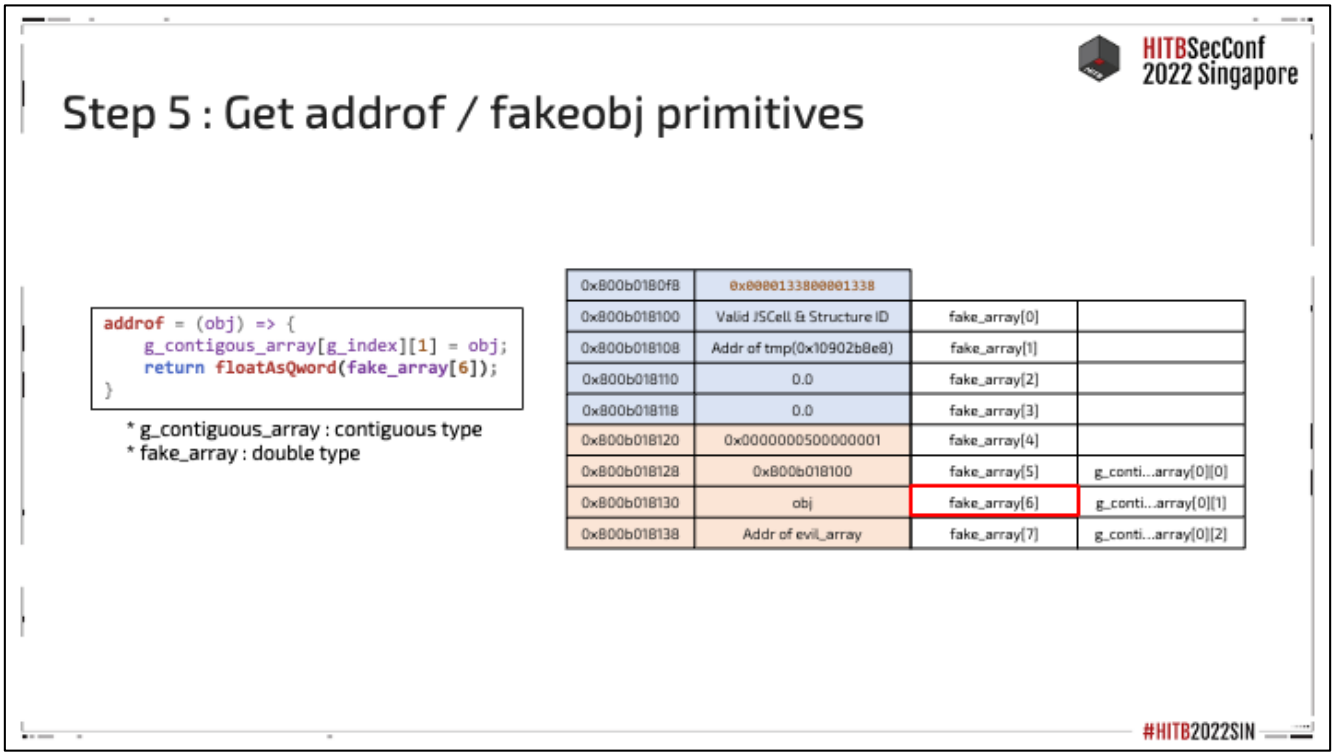


We are almost there…

Finally, we only need to explain how to make the addrof(어드레스오브) and fakeobj(페이크오브젝) primitives.
거의 다 왔다! 마지막으로 addrof와 fakeobj 메소드를 구성하는 방법만 설명하면 된다.

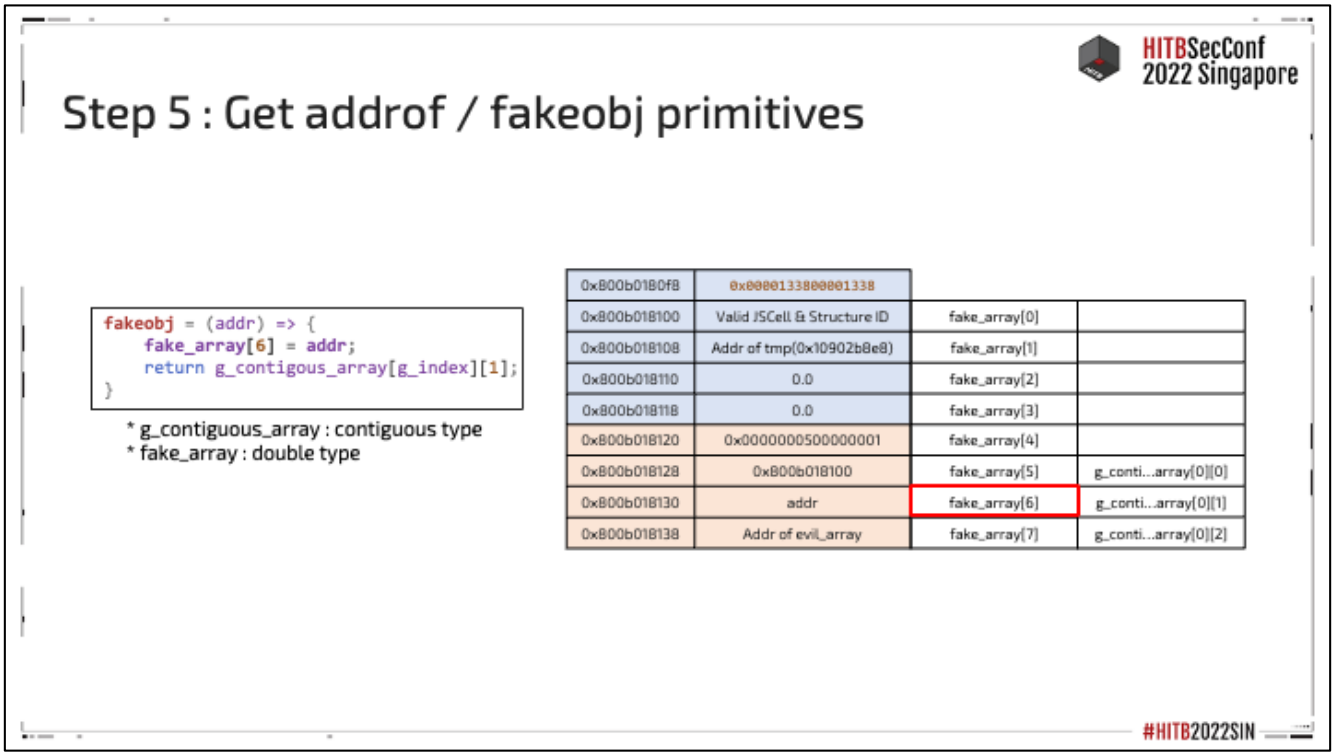
let's look at the addrof method.
addrof 메소드부터 알아보자.

Put the object we want to find out into the index 1(one) of g_contiguous_array.
주소를 알아내고 싶은 객체를 g_contiguous_array의 1번째 인덱스에 넣는다.



And, it read / using fake_array, as you know fake_array is double type array.
그리고, double type인 fake_array를 이용해 이를 읽어낸다.

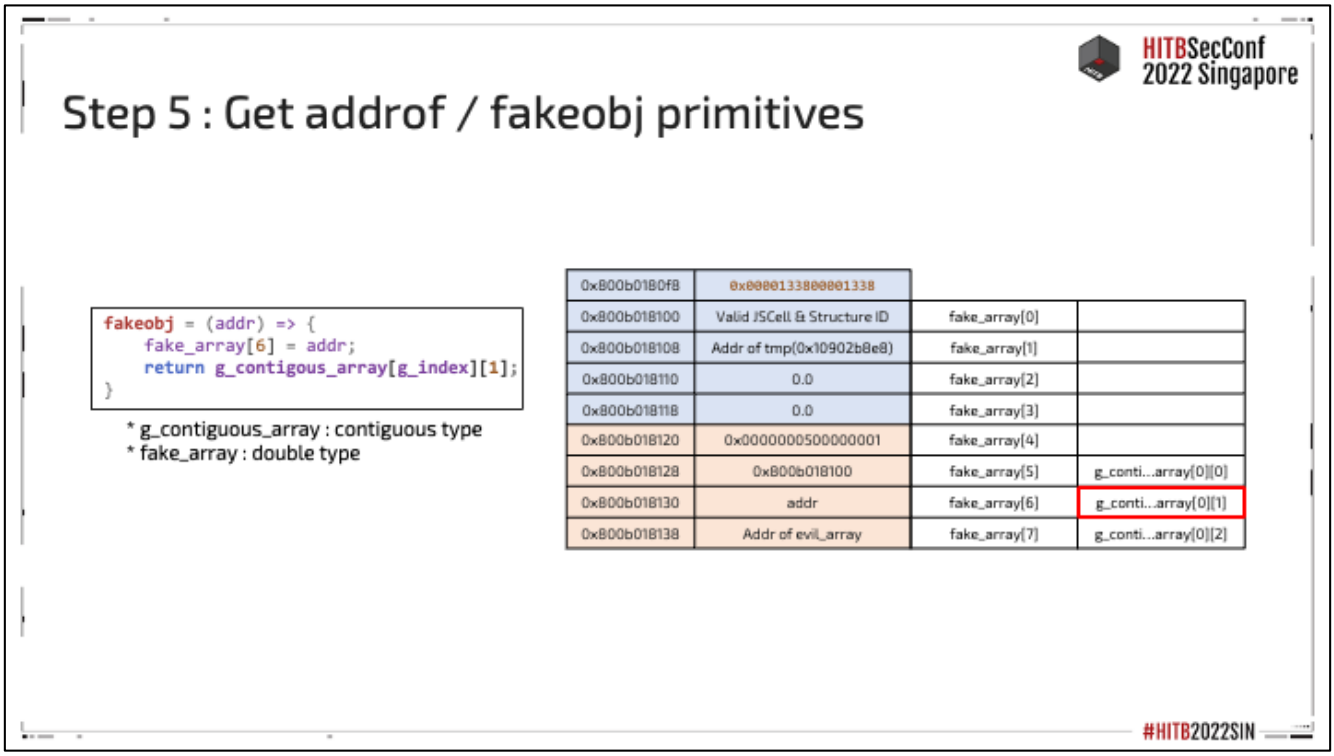
Then, you can get an object address in double format.
그러면, double 형식으로 출력된 객체 주소를 얻어올 수 있다.



Now, turn on the fakeobj code. Similar to the Addrof method, but a little bit different.
이제 fakeobj 코드이다. Addrof와 유사하지만, 조금 다르다.

First, we need to create a fake object in memory.
먼저 메모리에 가짜 객체를 생성해야 합니다.

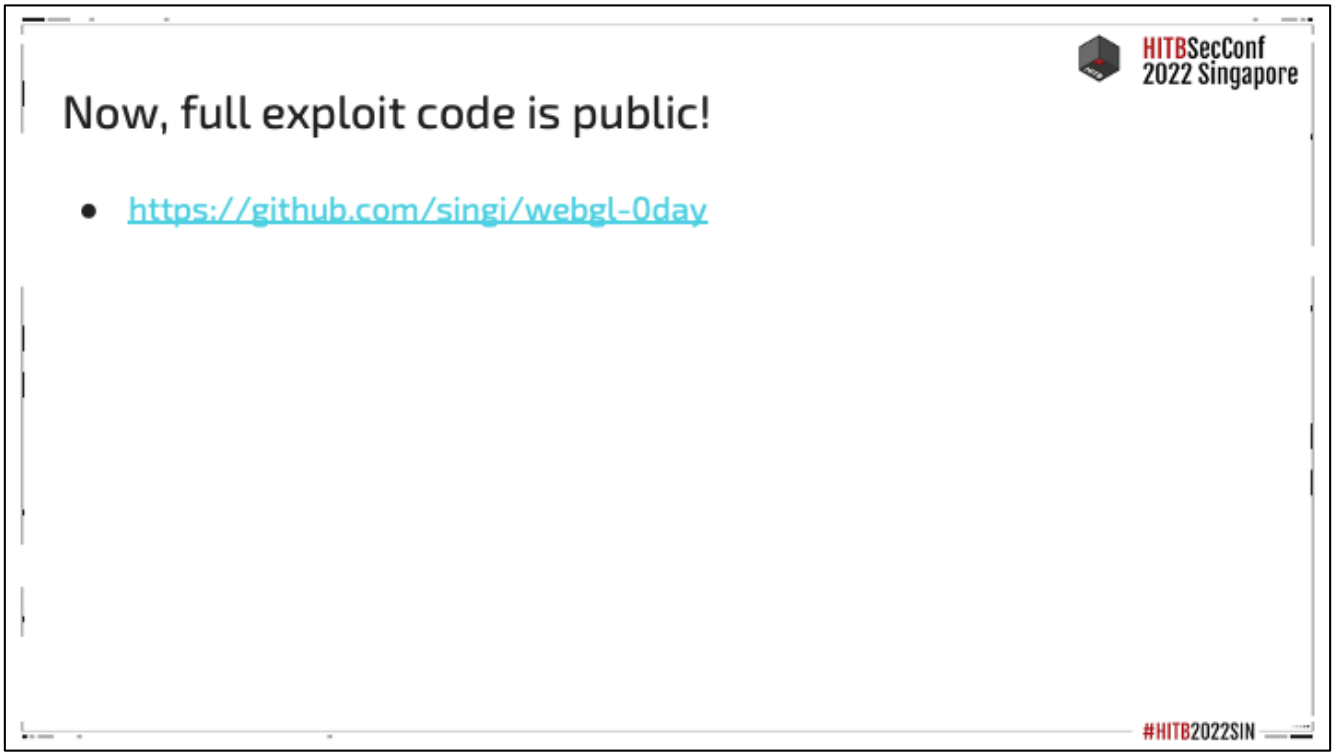
After that put the fake object address to the index 6th of fake_array.
그런 다음 fake_array의 인덱스 6번째에 가짜 개체 주소를 씁니다.



Now, read the index 1 of g_contiguous_array, the fake object pointed(포인티드) to by addr is recognized(레킥-나이즈드) as an object.
그리고, g_contiguous_array의 1번째 인덱스로 읽으면, addr이 가리키고 있는 fake object를 객체로 인식한다.

At this time, fakeobj must have a valid StructureID so that an(소뎃엔) / error does not occur(에러 더즈 낫 어컬).
이 때에 valid한 StructureID가 있어야 오류가 발생하지 않는다.

The core of addrof and fakeobj primitives / Store and load two arrays / of different types from / one share butterfly.
Addrof 와 fakeobj의 핵심은, 타입이 다른 2개의 배열의 store와 load를 1개의 공통 butterfly에서 하는 것이다.



I have posted / what we have learned so far / on Github.
지금까지 알아본 내용을 이 깃허브에 올려두었습니다.

And as a small solution for my English, I'm also posting a version of this slide with scripts.
그리고 제 영어에 관한 해결책으로 스크립트가 포함된 버전의 슬라이드도 함께 올립니다.

Now I switch to public.
이제 공개로 전환합니다.

Finally, let's take a look at the demo.
그리고 실제 데모를 한번 보시죠~!

check the browser version, and after that try to exploit.
버전 확인 후, exploit을 시도 합니다.

it is fail... it occurs to wrong predict butterfly address.
첫번째 실패 했네요. 이것은 예측한 butterfly 주소가 달라서 발생합니다.

and then, exploit is keep running until success.
성공할 때 까지, 재시작합니다.

if the "found array" message shown, our exploit is success.
found array라고 출력 되면 renderer 익스플로잇은 성공 한 것이다.

but, the exploit is not the full exploit chain, so I only changed to value of registers.
하지만, full exploit chain이 아니기 때문에 렌더러 프로세스의 레지스터 값만 바꿔봤다.

Thanks for listening to this talk. (지금까지 발표를 들어주셔서 감사합니다.)

If you have experienced(익스피리언스드) any trouble due to my English / please complain on Twitter DM. (영어 발음으로 인해 불편을 겪으셨다면, 트위터로 항의해주세요.)

Thanks.

“””

And at this time, I have to say, “do you have any questions?” like other speakers… (그리고 이 타이밍에 보통 Any question?이라고 물어봐야 합니다.)

I’ll listen carefully, to your question. Or using sildo, I would recommend this to save the time. Anything is OK. Let’s try it. (질문을 잘 들어볼게요.)

Any question?

If you have any further questions, please speak to me as I will be at the conference hall.

Thanks.

“””

References

- ANGLE: OpenGL on Vulkan (Jamie Madill, Google)
  - https://www.youtube.com/watch?v=QrIKdjmpmaA