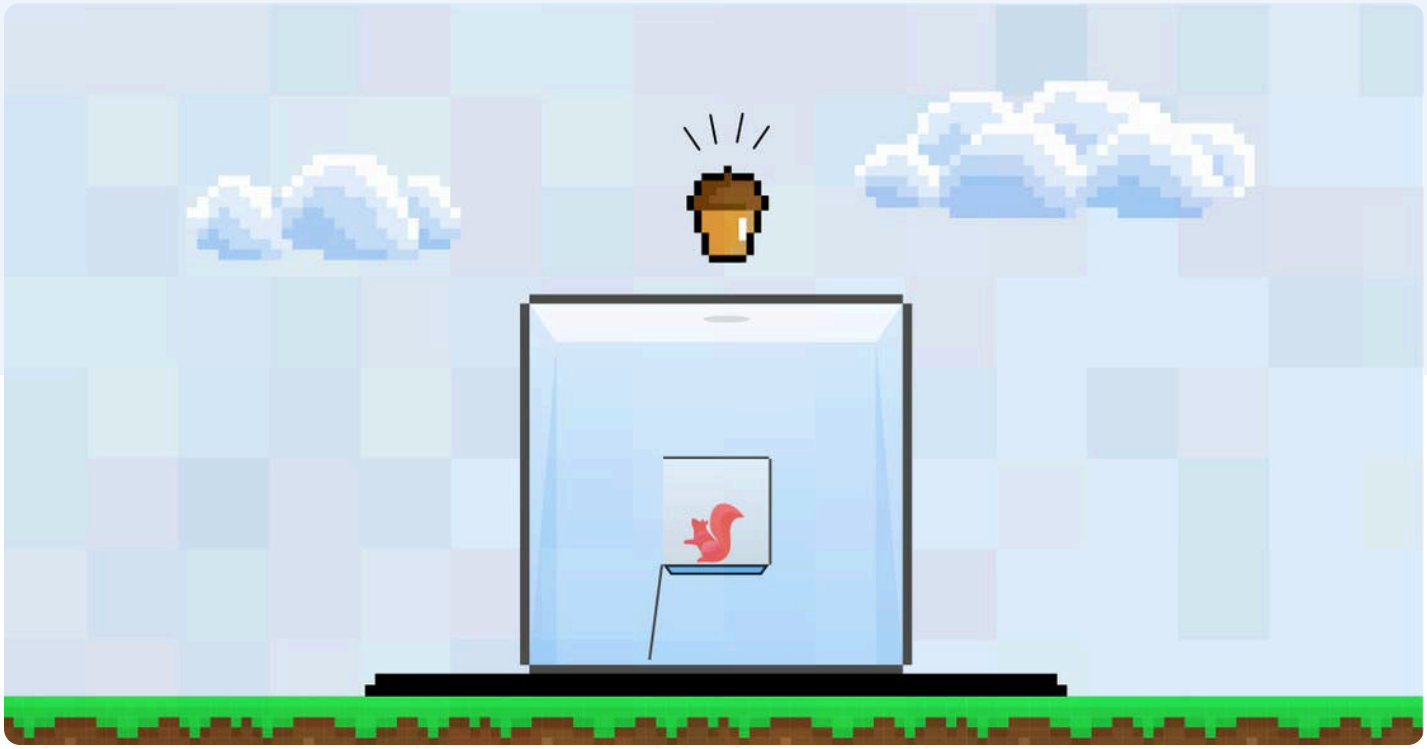
BLOG POST

Squirrel Sandbox Escape allows Code Execution in Games and Cloud Services

October 19, 2021



Simon Scannell and Niklas Breitfeld

VULNERABILITY RESEARCHERS

TL;DR overview

- Sonar's vulnerability research team discovered a sandbox escape vulnerability in the Squirrel scripting language virtual machine, a lightweight embeddable scripting engine used in game engines, IoT devices, and other applications that rely on sandboxed script execution.
- The vulnerability allows a malicious Squirrel script to escape the intended sandbox and execute arbitrary code in the host process, undermining the security model of applications that depend on Squirrel for safe execution of untrusted code.
- The finding demonstrates the risk of using scripting engines with complex language features—such as Squirrel's generators and delegates—without comprehensive security analysis of the runtime implementation.
- Responsible disclosure was followed; users of Squirrel-based applications should apply available patches and audit their use of untrusted script execution contexts.

SquirrelLang is an interpreted, open-source programming language that is used by video games and cloud services for customization and plugin development. For example, the extremely popular game Counter-Strike: Global Offensive (CS:GO) attracts millions of players on a monthly basis and utilizes the Squirrel Engine to enable anyone to create custom game modes and maps.

However, this freedom comes with a price: Anyone who downloads and hosts such an item from the community executes Squirrel code without any warning. Some of the most popular community-created items have been downloaded millions of times in the popular Steam shop. In order to prevent malicious actors from exploiting this, the Squirrel Engine is carefully sandboxed within the CS:GO process.

In this blog post, we break down a vulnerability we discovered in the core of Squirrel which was developed in C. It enables an attacker to bypass the

sandbox restrictions and execute arbitrary code within a SquirrelVM, giving the attacker full access to the underlying machine.

Impact

An attacker can exploit an Out-Of-Bounds Read vulnerability (CVE-2021-41556) to escape a Squirrel VM and gain access to the underlying machine. This attack vector becomes relevant when a Squirrel Engine is used to execute untrusted code. This is the case with cloud services such as, for example Twilio Electric Imp or video games such as Counter-Strike: Global Offensive and Portal 2 which attract millions of players monthly.

For example, in a real-world scenario, an attacker could embed a malicious Squirrel script into a community map and distribute it via the trusted Steam Workshop. When a server owner downloads and installs this malicious map onto his server, the Squirrel script is executed, escapes its VM, and takes control of the server machine. From here, as our recent research has shown, it would be possible to exploit other vulnerabilities within the game's network protocol stack that target the CS:GO players connecting to the hijacked server.

We verified that both stable release branches, 2.x and *3.x,* of Squirrel, are affected by the vulnerability discussed in this blog post. A patch has been released as a commit to the official Squirrel repository, but at the time of writing, this commit has not been included in a new stable release. The latest official release is from 2016 and does not include patches for numerous other vulnerabilities that have been reported over the years. We did not develop exploits for specific projects that use Squirrel, but we recommend all project owners who depend on Squirrel to rebuild the latest Squirrel version from source code.

Technical Details

In the following sections, we provide some background information necessary to understand this vulnerability. We then go into detail about the bug that led to this security issue and finally provide a high-level exploitation strategy.

Background - Squirrel Classes and Members

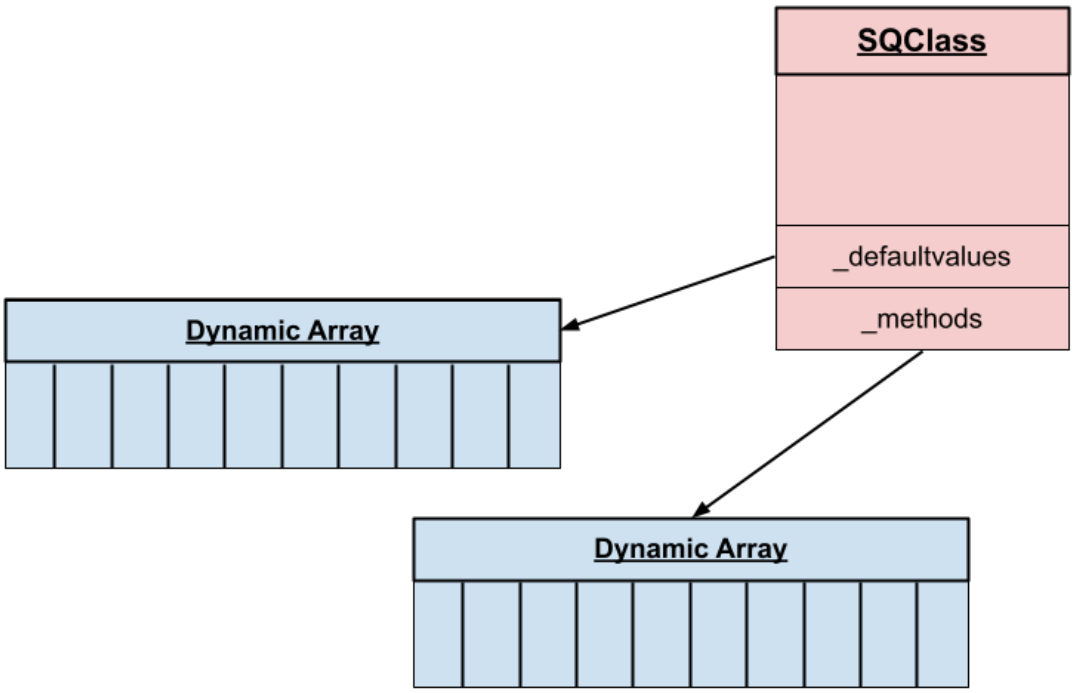
Squirrel is an object-oriented programming language similar to PHP. It allows developers to define classes and methods. To get a feel for Squirrel, let's assume the following example class definition:

```
class ExampleClass {
   first_field = 1;
   second_field = true;

   function someMethod() {
      print("Hello, World");
   }
}
```

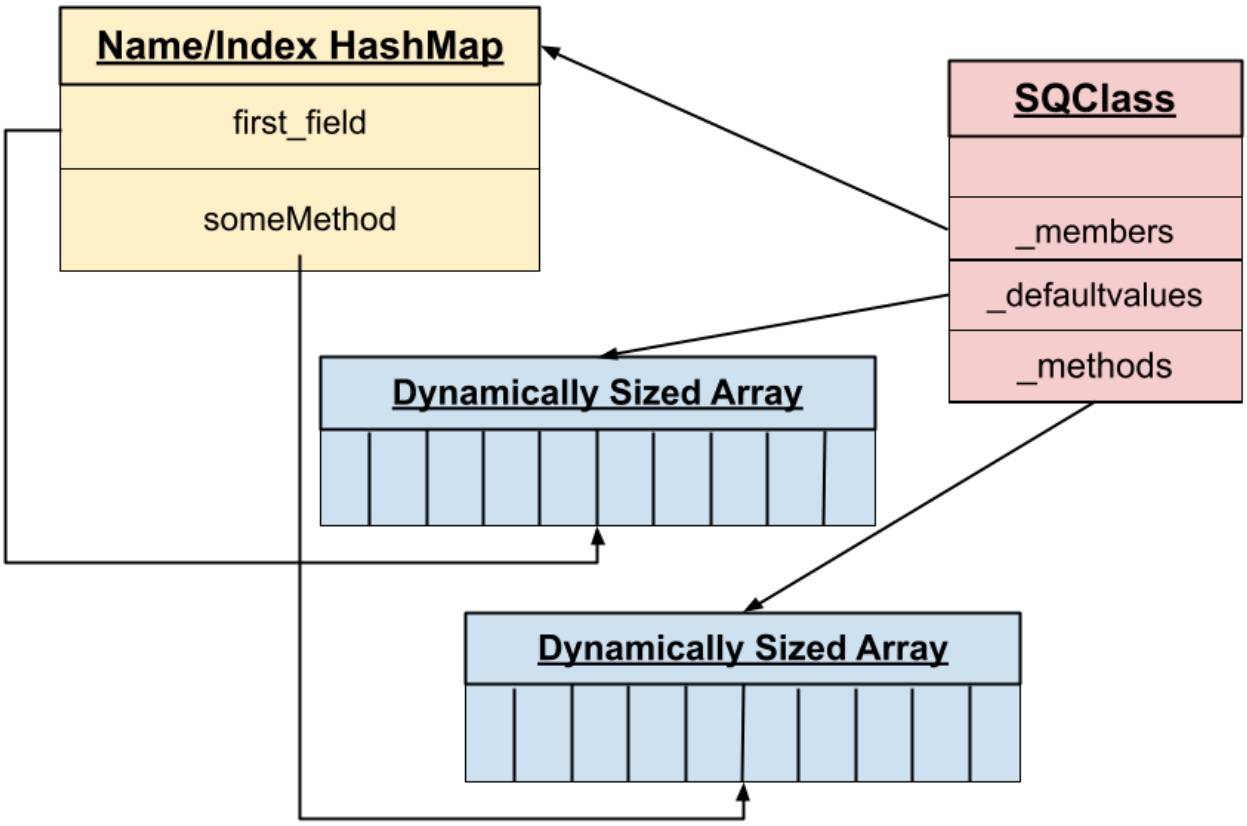
The syntax shown in the code snippet above is not very unique. It demonstrates a class definition with some default fields and a method named `someMethod()`.

In order to understand what is happening internally, let's look at internal structures and how they would appear in memory at runtime. On a simplified and abstract level, the underlying C code structures of this class definition could look like the following:



The above image shows how a Squirrel class definition (`SQClass`) contains a pointer to a dynamic array of methods, in which `someMethod()` would be stored, as well as a pointer to a dynamic array of default values.
Both `first_field` and `second_field`, along with their default values, would be stored here.

In order to access these default values and methods, an SQClass definition also contains a pointer to a HashMap. HashMap `_members` maps the name of attributes to their index within either the `_defaultvalues` or `_methods` array. This relationship is shown in the following graphic:



In order to determine if the retrieved index should be used to access the `_methods` or `_defaultvalues` array, a bitflag within the index is used.

The following code snippet shows a call to `_members->NewSlot()`, which is called when a class member is defined. We assume that a new default value, for example, `first_field`, is added to the `_members` HashMap:

squirrel/sqclass.cpp

```
53   bool SQClass::NewSlot(SQSharedState *ss,const SQObjectPtr &key,const
SQObjectPtr &val,bool bstatic)
54   {
...
94       SQClassMember m;
95       m.val = val;
96       _members->NewSlot(key, _make_field_idx(field_idx));
```

```
97      _defaultvalues.push_back(m);
98    return true;
99   }
```

Following the example of defining `first_field`, the `key` variable would contain the string "`first_field`". The corresponding `_defaultvalues` array index stored in the `field_idx` variable is then stored in the HashMap. Note, however, that before the index is stored it is modified with the `_make_field_idx()` macro. This macro and its counterpart for methods are defined as follows:

squirrel/sqclass.h

```
18   #define MEMBER_TYPE_METHOD 0x01000000
19   #define MEMBER_TYPE_FIELD 0x02000000
20
21   #define _ismethod(o) (o&MEMBER_TYPE_METHOD)
22   #define _isfield(o) (o&MEMBER_TYPE_FIELD)
23   #define _make_method_idx(i) ((MEMBER_TYPE_METHOD|i))
24   #define _make_field_idx(i) ((MEMBER_TYPE_FIELD|i))
25   #define _member_type(o) (o & 0xFF000000)
26   #define _member_idx(o) (o & 0x00FFFFFF)
```

As can be seen in line 24, the `_make_field_idx()` sets the bitflag `0x02000000` on the index.

CVE-2021-41556: Out-Of-Bounds Access via Index Confusion

The fact that bitflags are set within indexes is problematic as it is entirely possible for an attacker to create a class definition with `0x02000000` methods. As such we can create a very simple PoC:

```
class D {}
function a() {}
for(local i = 0; i < 0x02000008; i+=1) {
    D.rawset(i, a);
}
local xxx = D.rawget(0x02000004);
```

The `rawset` and `rawget` functions allow us to handily access members of a given class. In this PoC, the squirrel interpreter will dereference a null pointer and segfault because the `_defaultvalues` array has not been allocated yet.

The following code snippet shows the vulnerable code, which we will break down in the following paragraphs:

squirrel/sqclass.h

```
40   bool Get(const SQObjectPtr &key,SQObjectPtr &val) {
41      SQObjectPtr idx;
42      if(_members->Get(key, idx)) {
43         if(_isfield(index)) {
44            SQObjectPtr &o = _defaultvalues[_member_idx(idx)].val;
45            val = _realval(o);
46         }
47         else {
48            val = _methods[_member_idx(val)].val;
49         }
50         return true;
```

The above code is called when a class attribute is accessed. The call to `_members->Get(key, idx)` in line 42 takes in a key, which contains the name of the member that will be accessed. After the call, `idx` contains the index to either the `_defaultvalues` or `_methods` array. Which array should be accessed is determined by checking the bitflags of the index.

The `_isfield()` macro returns `true` if the bitflag `0x02000000` is set in the index. The bug lies in the fact that an attacker that is able to insert at least `0x02000000` methods into a class definition can force this check to return true since the bitflag would be set.

To make this more concrete, let's walk through an example of how an attacker can trigger this vulnerability:

1. The attacker creates a class definition with 0x02000005 methods and 0x1 fields

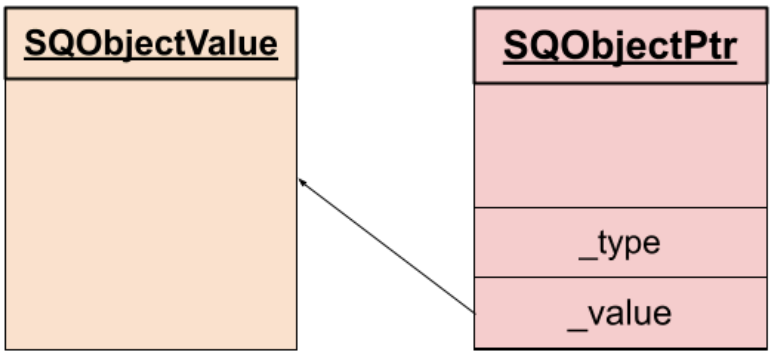
2. The attacker accesses the method with the corresponding index 0x02000005
3. The _isfield() macro returns true for this index as the bitflag 0x02000000 is set
4. The _defaultvalues array is accessed with index 0x5. However, it only contains 0x1 entries and thus the attacker has accessed out of bounds.

Exploitation Strategy

In order to understand why this out-of-bounds access is dangerous, let's have a look at what an attacker can do next.

The `_defaultvalues` array which is subject to the OOB-access contains `SQObjectPtr` structures. Thus, the memory that is read outside of the buffer of the array is interpreted as such. On a high level, this structure contains a pointer to a `SQObjectValue`, as well as a field that is used to determine what kind of object is referenced by the pointer.

The following graphic demonstrates the relationship between an `SQObjectPtr` and a `SQObjectValue`:



Through careful preparation of the heap, it is possible to craft a string that imitates an `SQObjectPtr` struct and place it next to the array of `_defaultvalues`.

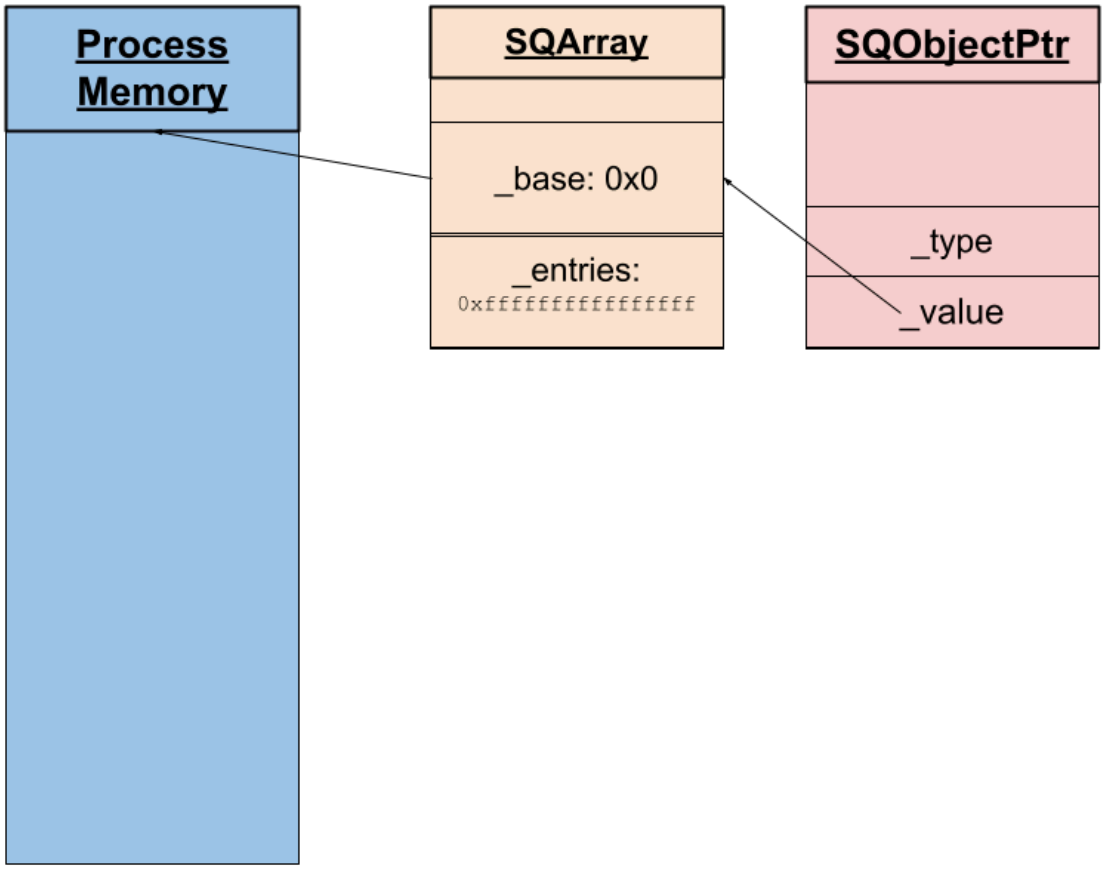
In the exploit we developed for demonstration purposes, we tricked the engine into believing that it fetched a pointer to a Squirrel Array. Squirrel Arrays are dynamic arrays, where 2 fields are relevant to exploitation:

1. A pointer that contains the address of the current array buffer
2. An 8-byte integer that contains the size of the current array buffer

By making the fake `SQObjectPtr` point to another attacker-controlled string on the heap, it was possible to trick Squirrel into returning an array that

points to the base address `0x0` and contains `0xffffffffffffffff` entries.

This enabled us to abuse the fake array to address the entire process space and read and write values. Ultimately, we were able to hijack the control flow of the program and gain full control of the Squirrel VM. This was achieved by overwriting function pointers. The following graphic shows this chain of attacker-controlled pointer that enabled reading and writing to the entire address space:



Timeline

Date	Action
2021-08-10	We send the vulnerability details via email to the email address listed in the Squirrel GitHub repository.
2021-08-12	We create a GitHub issue asking for the correct point of contact
2021-08-25	The maintainer replies with an email address we can disclose the vulnerability details to. We disclose the vulnerability details..
2021-08-26	The maintainer acknowledges the vulnerability.
2021-09-16	A commit containing a patch is pushed to the Squirrel GitHub repository by the maintainer.

Summary

In this blog post, we explained the details of an Out-Of-Bounds vulnerability in SquirrelLang. We outlined how storing information within an index value can lead to logical bugs if the bits representing this information are set too low. We also discussed how such issues might be exploited to escape a Squirrel VM and execute arbitrary code on a host process. We broke down how this might affect Counter-Strike: Global Offensive players to illustrate how such a vulnerability can be leveraged in the real world. We highly

recommend maintainers that are using Squirrel to apply the available fix commit to their projects to protect against these attacks. Last but not least, we would like to thank the Squirrel team for quickly making a patch available for this issue after our reporting.

Related Blog Posts

- Compilation database: An alternative way to configure your C or C++ analysis
- Ghost CMS 4.3.2 - Cross-Origin Admin Takeover
- elFinder - A Case Study of Web File Manager Vulnerabilities
- Zimbra 8.8.15 - Webmail Compromise via Email

SHARE THIS BLOG

Sonar