**NaCl Contest - Summary of findings**

Team CJETM

## Introduction

Team CJETM participated in the Google Native Client (NaCl) security contest in early 2009. This contest was held to find vulnerabilities in the NaCl software and reward the people who found those vulnerabilities. We have listed our findings below in accordance with the terms and conditions of the contest.

## Issues

### Issue 44 Uninitialized vtable may lead to arbitrary code execution

Version of Native Client: 0.1_38_2009_02_11
Browser Version Firefox 3.0.6
Operating System affected: Linux, Windows, OSX
http://code.google.com/p/nativeclient/issues/detail?id=44

Team CJETM discovered an uninitialized memory vulnerability through a manual source audit of the NaCl shared_memory.cc code. This vulnerability was the consequence of a number of issues including an integer overflow, not properly sanitizing memory returned by malloc and ignoring the return value of a critical constructor function. The vulnerability exists in the *SharedMemory::New* function. This function receives an attacker controlled length value, which is overflowed and turned negative after being casted to an *off_t* type. After allocating some space on the heap for an *imc_desc* structure this function calls the *NaClDescImcShmCtor* function which is supposed to initialize the *imc_desc* structures vtable, but *NaClDescImcShmCtor* returns 0 prematurely indicating an error because of the attacker controlled length value. The caller, *SharedMemory::New*, does not check for this error return value and proceeds to call another function *New* that calls a function pointer in the uninitialized *imc_desc* vtable. An attacker can exploit this issue by using a technique called heap spraying, this would fill the heap with the proper executable code to take advantage of the call to the uninitialized vtable function pointer.

Here is our proof of concept code to trigger the vulnerability:

```
<html><head><title>Go Boom</title></head>
<script type="application/x-javascript">
   var cjetm = function() {
     // heap spray would go here
     var server = document.getElementById('nacl_server');
     var socket_address2 = server.start_server();
     var con_sock = socket_address2.connect();
     alert('about to boom!');
     var region_size = 2147483648;
     var shared_memory = server.__shmFactory(region_size);
   }
</script><body>
<embed type="application/x-nacl-srpc" id="nacl_server" width="0" height="0"
src="srpc_nrd_server.nexe" />
<button type="button" onclick="cjetm();">Crash me</button></body></html>
```

After Google patched this vulnerability we noticed a change in the entire code base of NaCl. More specifically every constructor function that initialized an object with a vtable was specifically modified to eliminate the possibility of this vulnerability existing anywhere else.

**Issue 42 Unhandled Exception In new Operator**
(covers all three new operators fixed as a result of this vulnerability, including the integer overflow that leads to too small of an allocation which may result in a heap overflow)

Version of Native Client: 0.1_38_2009_02_11
Browser Version: Firefox 3.0.6
Operating System affected: Linux, Windows, OSX
http://code.google.com/p/nativeclient/issues/detail?id=42

Team CJETM discovered three vulnerable uses of the C++ new operator on line 584 of npapi_plugin/srpc/srpc_client.cc. Each new operator takes an attacker supplied integer as its length argument allowing an attacker to create denial of service or integer overflow situations when allocating memory. In the case of the integer overflow, too small of an allocation will be returned which may result in a heap overflow, which would allow for arbitrary code execution. This vulnerability was discovered via manually fuzzing different values into the example Javascript provided by the NaCl project.

The first vulnerable call to *new* allocates a character array and the other two are of types *integer* and *double*. Each of these three allocations can be reached by changing the value of the type of argument provided in *outputs[i]->tag*. These are all reachable by an attacker to create denial of service conditions by forcing the new operator to throw an unhandled exception. However in the case of the integer array this issue may be exploitable by triggering an integer overflow using the value 1073741824 which creates a smaller allocation than expected, which may result in a heap overflow when memory is copied into the buffer. Team CJETM considers these three separate vulnerabilities but they are reported as one (issue #42) during the contest.

The first vulnerable character array allocation can be easily demonstrated by supplying a negative integer to the *getmsg()* Javascript method exposed by the srpc_nrd_server.nexe executable provided by the NaCl project. The negative value is passed to the *new* operator where it is cast as an *unsigned int* and wraps to a large number that the underlying allocator cannot allocate. In our proof-of-concept code the C++ runtime throws an exception and Firefox exits.

Here is our proof of concept code that will trigger the vulnerability. It requires that the srpc_nrd_server.nexe executable is present. This nexe is distributed with the NaCl project code.

```
<html><head><title>Go Boom</title></head>
<script type="application/x-javascript">

var SocketAddressNaClToBrowser = function() {
   var server = document.getElementById('nacl_server');
   var socket_address2 = server.start_server();
   var con_sock = socket_address2.connect();
   alert('about to boom!');
   var retval = con_sock.getmsg(-1);
}
```

```
</script><body>

<embed type="application/x-nacl-srpc" id="nacl_server" width="0" height="0"
src="srpc_nrd_server.nexe" />
<button type="button" onclick="SocketAddressNaClToBrowser();">Crash
me</button></body></html>
```

After Google patched this vulnerability we noticed a change in the entire code base of NaCl. Many allocations created with the new operator have had *std::nothrow* added to them, this tells the new operator not to throw an exception when an allocation fails and instead allows the NaCl itself to catch the NULL pointer returned by *new*.

**Issue 70 Double Delete in plugin.cc**

Affected OS: Linux, OSX, Windows
Tested Browser: Firefox 3.0.6
NaCl Version: nacl_linux_0.1_57_2009_03_30.tgz
http://code.google.com/p/nativeclient/issues/detail?id=70

Team CJETM discovered a 'double delete' condition on line 308 of plugin.cc in the function *Plugin::Invoke*. This vulnerability was found by a manual source code audit, and required some knowledge of how the NP API functions.

In the vulnerable function NaCl sets up a *UrlAsNaClDescNotify* class instance named *callback*. This class instance is used to hold the URL passed to the *__urlAsNaclDesc* function in our javascript POC code. This URL is eventually passed to Firefox via the *NPN_GetURLNotify* function, along with a callback function pointer *NPP_URLNotify* in srpc.cc, for validation. By supplying an invalid URL the NP API will return error, the callback function will delete the class instance and also return error. When this happens *Plugin::Invoke* will also delete the class instance. A proof of concept for this issue can be easily created by modifying the URL passed to *__urlAsNaClDesc* in file srpc/srpc_url_as_nacl_desc.html included with the NaCl source code.

```
...
var url = 'http://http://www.google.com'; // Invalid URL
plugin.__urlAsNaClDesc(url, new FileLoadCallback(0, 1));
...
```

Exploiting this vulnerability is going to vary from platform to platform and by heap implementation. At the very least this issue will cause the browser to crash. This issue becomes exploitable if the attacker can control the contents of the deleted memory after the first call to *delete*. At this point he can place fake heap chunk data or more interestingly overwrite the original class' vtable that held the dtor address with an address that holds attacker supplied code that will be called when the second *delete* occurs. Even if the memory is not controllable by the attacker between each call to *delete* there is still a risk of the memory being added to a list of free chunks twice, and then being returned twice to two different memory allocation requests. The latter is less likely because of the dtor function being called after the second call to *delete* which will likely result in a crash if the attacker has not overwritten the data.

A casual look at the *SetProperty* function in plugin.cc also appears to follow the same code pattern, and may be vulnerable as well.

**Issue 49 SetWindow doesn't set window_**

Affected OS: Linux, OSX, Windows
Tested Browser: Firefox 3.0.6
Nacl Version nacl_linux_0.1_38_2009_02_11.tgz
http://code.google.com/p/nativeclient/issues/detail?id=49

Team CJETM discovered a vulnerability in the function *NPModule::SetWindow*. This function takes a structure named *window*, this structure holds height and width parameters which are supplied by an attacker using HTML as shown below:

```
...
<html>
  <embed id="embed1" src="npapi_test.nexe" type="application/x-nacl-npapi"
width="24576" height="24576" />
</html>
...
```

In the *SetWindow* function the height and width parameters are multiplied by each other and 4 in order to get a size which is assigned to *bitmap_size_* (this multiplication is sufficient enough for integer overflow but Firefox appeared to be limiting each value to max of 32768). This value is first passed to the function *CreateMemoryObject*. This function returns error due to a failed call to *ftruncate*, but its return value and the original *bitmap_size_* value are then passed to the *CreateShmDesc* function. This causes *CreateShmDesc* to return NULL on error before setting the modules *window_* structure to the structure initially passed to *SetWindow*. Eventually a call to *XtDispatchEventToWidget* occurs which calls the function *EventHandler* and eventually crashes due to a *window_->window* NULL pointer access. Each NaCl supported platform (Linux, Windows, OSX) have different *EventHandler* functions, so exploitation will vary between them. However a NULL pointer dereference and a crash are easy to trigger on each platform.

**Conclusion**

We regret that we were not able to spend more time on the contest. Each time we reviewed specific portions of the NaCl source code we were able to find critical vulnerabilities in it, particularly in the code that sets up the plugin and interfaces with the NP API. We concentrated our efforts by fuzzing inputs where appropriate and reviewing source code where we saw potential flaws.

*Team CJETM is Chris Rohlf, Jason Carpenter and Eric Monti*