现代的web浏览器提供了如ActiveX和NPAPI这样的扩展机制，使得本地代码能作为web应用程序的一部分被加载和运行，但这样对主机来说存在安全隐患。Google Native Client（NaCl），类似于微软的ActiveX技术，它运用沙盒技术，可以让浏览器直接运行机器码，让web应用程序得到更佳的性能，同時又兼顾安全性。一个NaCl应用程序由许多可信和不可信NaCl模块组成，每个模块都在一个进程中单独运行。Native Client至少应当具备在处理不可信的模块时有跟Javascript一样的安全性。概念上来讲，NaCl由两部分组成：对本地代码严格的执行环境和一个用来托管本地代码扩展的运行时。

NaCl系统的各个组成部分的详细设计

1.内外两层沙箱机制及其实现

 Native Client基于x86进程内“内层沙箱”构建而成。而为了进一步防御攻击，还构建了一个“外层沙箱”用于进程边界的隔离。

内层沙箱使用代码静态分析技术来在不可信的x86代码中检查安全隐患。以前这样的分析受到了很多技术的挑战，比如说可自我修改的代码，重叠指令。为了解决这样的问题，在Native Client中为代码制定了一系列契约和结构规则，从而确保本地代码模块能可靠地进行分解，然后通过代码验证器运用机器码验证器 (Code Validator)来确保可执行文件只包含合法指令集中的指令，即NaCl通过机器码验证器能保证只有安全的机器码才能在系统执行。由于x86/x86-64是[复杂指令集](http://zh.wikipedia.org/wiki/複雜指令集)，[指令](http://zh.wikipedia.org/wiki/指令)长度不一，在[控制流程](http://zh.wikipedia.org/wiki/控制流程)中可能隐藏非安全机器码，从而使验证十分耗时。NaCl运用固定长度的16或32[位](http://zh.wikipedia.org/wiki/字节)的指令束(Instruction Bundle)，使机器码验证器设计简单(只是约600行[C语言](http://zh.wikipedia.org/wiki/C語言))和高效率； 而运用指令束只会失去5%的执行效率。

内层沙箱用于在一个本地进程中创建一个安全的子域。在这个子域里我们可以将一个可信的运行时服务(Service Runtime)子系统和不可信模块放置在同一个进程中。通过跳转机制允许可信代码和不可信代码之间的控制转移。内层沙箱不仅将系统与本地模块进行分离，而且还使得本地模块与操作系统进行了分离。

本层沙箱的设计仅限于机器码明显的控制流（通常以调用和跳转形式出现）。其他类型的控制流，如异常，则由服务运行时管理。内层沙箱定义了一系列用于可靠分解的规则，一个用于监视这些规则的编译工具链，以及一个确保代码复合规则的静态代码分析器。这样的设计使得可信代码段（Trusted Code Base）因其中不包含汇编工具而体积很小，同时验证器也能在允许通过测试和审查的时候做到体积也足够小。验证器的实现只需要少于600条的语句（分号），包括X86解码器实现和cpuid解码工作。编译后大约有6K字节的可执行代码（Linux 优化编译情况下），这其中大约有900字节的cpuid实现，1700字节的解码器实现和3400字节的验证器逻辑。为了消除由此带来的副作用，验证器必须解决四个子问题：

数据完整性：在数据沙箱外没有对数据的加载和存储

可靠的分解

没有不安全的指令

控制流完整性

为了解决这些问题，NaCl在已有的关于CISC错误隔离的基础上建立了新的方式：把80386的分段内存技术和CISC软件错误隔离技术相结合。我们使用80386段来限制在虚拟32位地址空间中的连续子段中的数据引用，这样就能高效地实现一个不需要加载和存储指令的数据沙箱。值得注意的是NaCl模块是32位的x86可执行模块，不支持64位。Native Client对于不可信二进制代码有7条规则限制（C1-C7），如下表。总的来说，C1和C6使得代码分解是可靠的。

C1 Once loaded into the memory, the binary is not writable,

enforced by OS-level protection mechanisms during execution.

C2 The binary is statically linked at a start address of zero,with the first byte of text at 64K.

C3 All indirect control transfers use a nacljmp pseudo- instruction (defined below).

C4 The binary is padded up to the nearest page with at

one hlt instruction (0xf4).

C5 The binary contains no instructions or pseudo-instructions overlapping a 32-byte boundary.

C6 All valid instruction addresses are reachable by a fall- through disassembly that starts at the load (base) address.

C7 All direct control transfers target valid instructions.

NaCl禁止一些操作码的使用,比如syscall和int（不可信代码就无法直接调用系统调用）、所有修改x86段状态的指令（如lds等）以及ret指令。Native Client允许hlt指令，但它会导致模块立即终结。处于安全考虑，还禁止所有其他的特权0级的指令，因为在一个正确的用户态指令流中不该使用。Native Client 也限制使用x86前缀，这样就只允许已知有用的指令运行。从经验上来讲，我们发现这样做消除了与CPU勘误表有关的一定的拒绝服务漏洞。

第四个问题是控制流的完整性，这样能保证所有在程序文本里的控制转移都在代码分解时有了相应的已识别的指令。

外层沙箱用于限制普通的系统调用，使得本地程序只能通过服务运行时来进行系统调用。若内层沙箱被攻破，不可信代码就可以进行系统调用，但此时外层沙箱可以控制进程的系统调用来阻止危险行为。Linux 上的实现使用ptrace 接口， 而Windows 上使用访问控制列表。Linux 上的实现将NaCl 容器作为一个子进程启动，然后使用ptrace 对进程发出的所有系统调用进行检查。外层沙箱维护一个允许使用的系统调用白名单，试图执行任何白名单以外的系统调用都会导致NaCl 模块立即结束。

2.运行时机制及其实现

Native Client为进程间通信提供了一个可靠的数据报抽象机制——IMC服务或直接称为IMC。IMC允许可信和不可信模块之间越过进程边界发送/接收包含无类型字节数组的数据包，还可以包含用于文件共享、共享内存对象及通信通道的“NaCl资源描述符”。IMC同时也是两个更高级的抽象机制——SRPC和NPAPI的基础。第一种SRPC，它用于定义和使用子过程来跨模块边界访问，包括从浏览器中的JavaScript调用NaCl代码。第二种NPAPI，它提供了与浏览器状态交互的接口，包括打开URL和对DOM的访问。该接口遵循已有的关于内容安全方面的规则。这两种机制的任一机制都可以用来与浏览器进行交互，包含页面内容修改、处理鼠标和键盘事件和获取其他站点内容，因为这些都可以通过JavaScript访问。

      如上所说，运行时服务负责为NaCl模块之间，以及和浏览器之间的交互提供容器。它一般会提供应用程序编程环境相适应的一系列系统服务。为了支持malloc/free接口或其他内存分配抽象机制，它提供了sbrk()和mmap()系统调用。它提供了POSIX线程接口的一个子集，用于线程的创建和销毁、条件变量、互斥量、信号量及线程本地存储。它还提供了常用的POSIX文件I/O接口，用于通信通道以及基于web的只读内容的操作。为了防止恶意的网络访问，connect()和accept()这样的系统调用不可使用。

服务运行时是一个本地可执行代码，它被一个NPAPI插件调用，这个插件也用来支持服务运行时和浏览器之间的交互。它实现了dynamic enforcement机制，这个机制维护了内层沙箱的完整性，并提供了一个资源抽象，这个资源抽象用来把NaCl应用程序从主机资源和操作系统接口中分离出来。它包含了可信的代码和数据，但与包含的NaCl模块共享同一个进程，且后者可以通过一个受控的接口去访问服务运行时。服务运行时通过x86的内存分段和分页机制相结合的方式来阻止不可信代码的非法内存访问行为。

      当一个NaCl模块被加载进来，它就被置于一个服务运行时地址空间中段分隔的256M区域内。NaCl模块地址空间（NaCl“用户”空间）的前64Kb被服务运行时保留，用于初始化。前4kb受读写保护，用于检查空指针。剩下的60Kb包含了实现“间接跳转”调用门和“记分板”返回门的可信代码段。不可信的NaCl模块文本紧随64kb后的区域加载进来。CS段寄存器用来约束从0基址到NaCl模块文本的末尾的控制转移，其它的段寄存器则用来限制对于256MB大小的NaCl模块地址空间的数据访问。

由于间接跳转和计分板代码是由可信服务运行时产生和安装的，所以它们可以包含在不可信的可执行文本里的其它任何地方都被禁止的指令。这个代码因为是NaCl模块加载进程的一部分而被作为运行时补丁，它通过使用段寄存器操作指令和远程调用指令实现在不可信的用户代码和可信的服务运行时代码间的控制转移。NaCl用户地址空间里的前64KB地址中，由于每个32的整数倍的地址都可能是计算后得出的控制流目标，所以这些地址就作为我们的系统调用间接跳转表的进入点。这些进入点中有一个被用hlt指令阻塞，这样剩余的地址空间就可以用于那些只能被服务运行时调用的代码。这样做为计分板返回门提供了空间。

对一个间接跳转的调用会实现从不信任代码到可信代码的控制权转移。间接跳转序列重置ds段寄存器，然后通过重置cs段寄存器（通过远程调用实现）和重置对于可信的服务处理者的控制转移来重新建立直接寻址模式，而这正是服务运行时所期望的。一旦处于NaCl用户地址空间外，它会重置其它的段寄存器，如fs、gs及ss，并为这个线程完全地关闭内层沙箱，通过这种方式重建一个跟本地代码一样的线程环境。服务运行时用一个能用的可信的栈地址来加载栈寄存器esp。 每个线程的可信的栈的地址空间都在不可信的地址空间外，这样，不可信的NaCl模块中的线程就不会攻击到这些栈。

正如间接转移允许不可信代码跨入可信代码一样，计分板从另外的方向也允许了这种跨越。可信的运行时用计分板来控制转移给任一的不可信地址、启动一个新的POSIX标准的线程和启动主线程。对齐机制（Alignment）保证了计分板不能直接被不可信的代码调用。

3.通信

IMC 是NaCl 模块与外界通信的基础。它基于NaCl socket 实现，提供了一个

类似Unix domain socket 的双向、可靠和有序的数据包服务。当一个不可信的NaCl

模块被创建时（通过DOM对象创建），它会接受到第一个NaCl socket。这个NaCl模块可以通过JavaScript访问。JavaScript 使用这个socket 向NaCl 程序发送消息，也可以将其共享给其他NaCl模块。JavaScript 还可以将NaCl socket 共享为NaCl 描述符，从而选择将模块连接到其他服务。NaCl 描述符还可以用来创建共享的内存段。

通过使用NaCl 消息，NaCl 的SRPC抽象 完全在不可信代码中实现。SRPC为定义在JavaScript和NaCl模块间，或是两个NaCl模块间的过程接口提供了便捷的语法。

这种语法支持一些基本的类型（int，float，char）、数组和NaCl 描述符，不支持指针。

外部数据表示法例如XDR或者protocol buffer能很容易地应用于NaCl消息或者SRPC之上。NPAPI的实现也在IMC上，并且支持部分的常用NPAPI接口。为实现当前的NPAPI，具体需求有：读、修改和调用浏览器中的脚本的属性和方法的能力 、对简单光栅图形技术的支持、提供createArray()方法和把URL能像文件描述符一样打开和使用的能力。尽管当提升了我们的相关安全考虑和应用需求后，我们很可能会进一步地约束和扩展NPAPI，但作为权宜之计，我们还是首选当前已实现的NPAPI子集。

4. **异常**

      不允许使用硬件异常（段错误，浮点数异常）和外部中断。为了隔离异常，每个NaCl模块都在一个单独进程中运行，无法使用异常处理来从硬件异常中恢复过来，而是这个进程会被关闭。虽然不支持硬件异常，但确实支持c++异常，但不支持Windows结构化异常处理(WSEH)。

5.开发者工具

编译NaCl模块：修改了标准GNU工具链，采用v4.2.2 gcc编译器和v2.18 binutils来生成NaCl规则二进制文件。用这个修改了的GNU工具链，我们已编译了一个newlib二进制文件。NaCl支持一个不安全的debug模式，这个模式允许附加的文件系统交互操作。

修改了gcc函数，使得函数项以 (-falign-functions) 32字节对齐。为使用间接的控制转移nacljmp指令，在gcc里加入了间接调用和所有的返回。还通过扩展nacljmp伪指令，修改了汇编语言用于实现间接控制转移序列，同时也修改了连接器，用以设置NaCl加载器所需的镜像基地址（64K）。