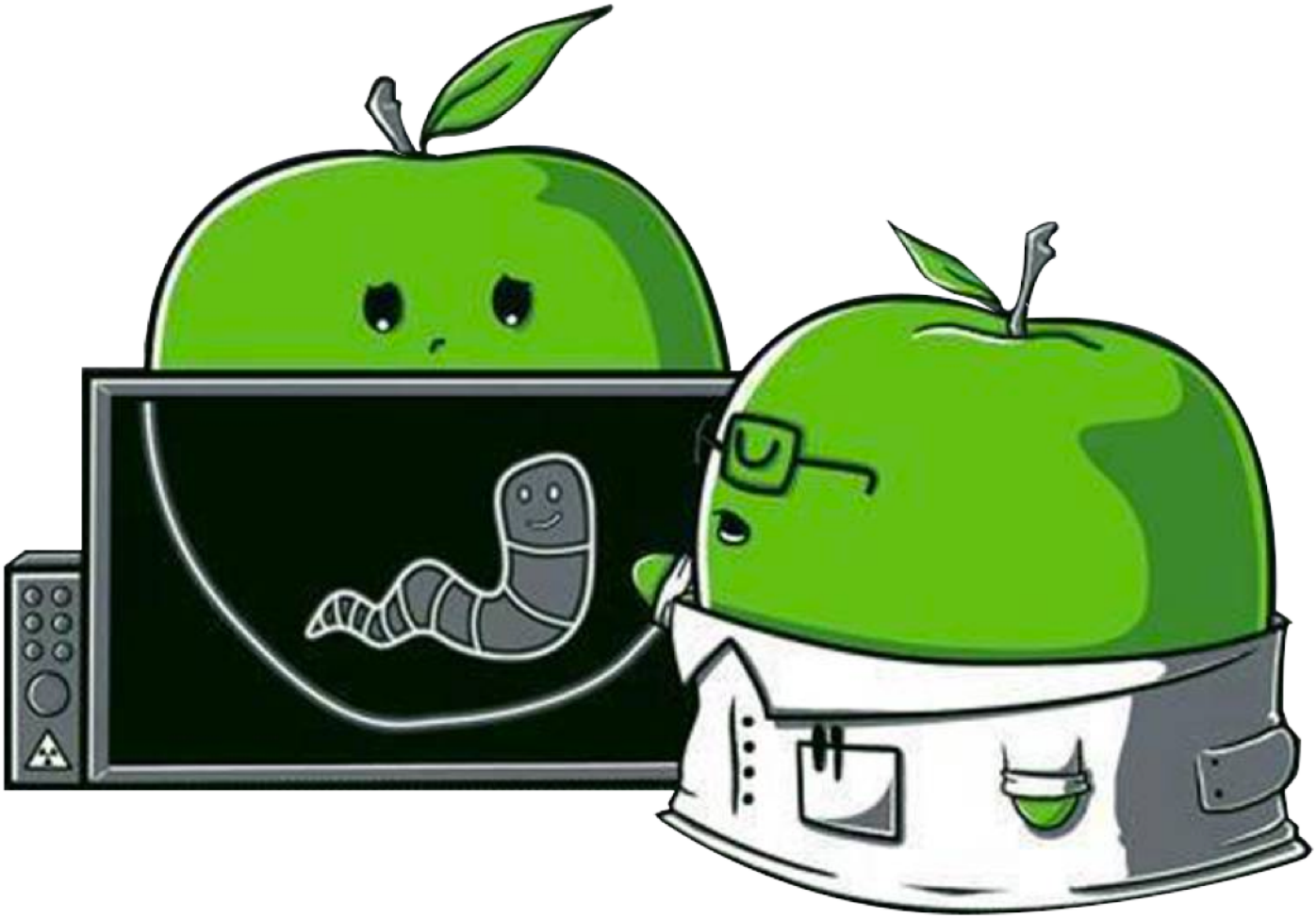


第0x0A章：调试

笔记:

这本书正在进行中。

我们鼓励您直接在这些页面上发表评论 ...建议编辑、更正和/或其他内容！

要发表评论，只需突出显示任何内容，然后单击 出现在屏幕上的图标
就在文档的边界上）。

在前一章中，我们介绍了被动动态分析工具，包括进程、文件和网络监视器。而这些工具通常可以（很快！）提供对恶意样本的宝贵洞察，但有时他们无法。此外，虽然它们可能允许人们观察受审查样本的行为，但此类观察是间接的，因此可能无法提供对样本内部工作的真实见解。需要更强大的东西！

最终的动态分析工具是调试器。简单地说，调试器允许一条一条地执行二进制指令。任何时候都可以检查（或修改）寄存器和内存内容，跳过（绕过）整个函数，等等。

在深入研究调试概念之前，先来看一个简单的例子 ...这个例子清楚地说明了调试器的功能。奥斯。Mami [1]包含大量嵌入的加密数据，并将其传递给名为setDefaultConfiguration的方法：

```
[SBConfigManager设置默认配置:
@"uZmgulcipekSbayTO9ByamTUu_zVtsflazc2Nsuqgq0dXkoOzKMJMNTULoLpd-QV9qQy6VRluzRXqWOG
scgheRvikLkPRzs1pJbey2QdaUSXUZCX-UNERrosul22NsW2vYpS7HQO4VG5l8qic3rSH_fAhxsBXpEe55
7eHIr245LUYcEIpemnvSPTZ_lNp2XwyOJjzcJWirKbKwtc3Q61pD..."];
```

一般来说，恶意样本中的加密数据是恶意软件作者试图隐藏的数据 ...要么来自检测工具，要么来自恶意软件分析师。作为后者，我们当然非常有动力解密这些数据，以揭示其隐藏的秘密。

就OSX而言。Mami，基于上下文（即调用名为setDefaultConfiguration:）的方法），似乎可以合理地假设该嵌入数据是恶意软件的（初始）配置，其中可能包含有价值的信息，如命令和控制服务器的地址、对恶意软件功能的了解等。

那么如何解密呢？静态分析方法将非常缓慢且效率低下，而文件或进程监视器将几乎没有用处，因为加密的配置信息既不会写入磁盘，也不会传递给任何（其他）进程。换句话说，它只存在于内存中，被解密。

通过调试器（稍后将介绍更多），我们可以指示恶意软件执行，并在SBConfigManager的setDefaultConfiguration:方法处停止。然后，一条指令一条指令地“步进”（执行），我们允许恶意软件以受控的方式继续执行，在完成对其配置信息的解密后再次暂停。

由于调试器可以直接检查正在调试的进程的内存，因此我们可以简单地“转储”（打印）现在已解密的配置信息（由rax寄存器指向）：

```
# lldb MaMi
（lldb）目标创建“MaMi”
当前可执行文件设置为“MaMi”（x86_64）。

...

(lldb) po $rax
{
 "dnsChanger" = {
   "affiliate" = "";
   "blacklist_dns" = ();
   "encrypt" = true;
   "external_id" = 0;
   "product_name" = dnsChanger;
   "publisher_id" = 0;

    ...

   "setup_dns" =         (
      "82.163.143.135",
      "82.163.142.137"
    );

    “共享存储”=“/Users/%USER\u NAME%/Library/Application Support”；“存储
    超时”=120；
   };

 "installer_id" = 1359747970602718687;
 ...
}
```

各种解密的密钥/值对（如“product_name”=Dnshanger）提供了对恶意软件最终目标的洞察；劫持受感染的系统DNS设置，迫使域名解析通过攻击者控制的服务器路由（如解密配置中所述，在82.163.143.135和82.163.142.137处找到）。

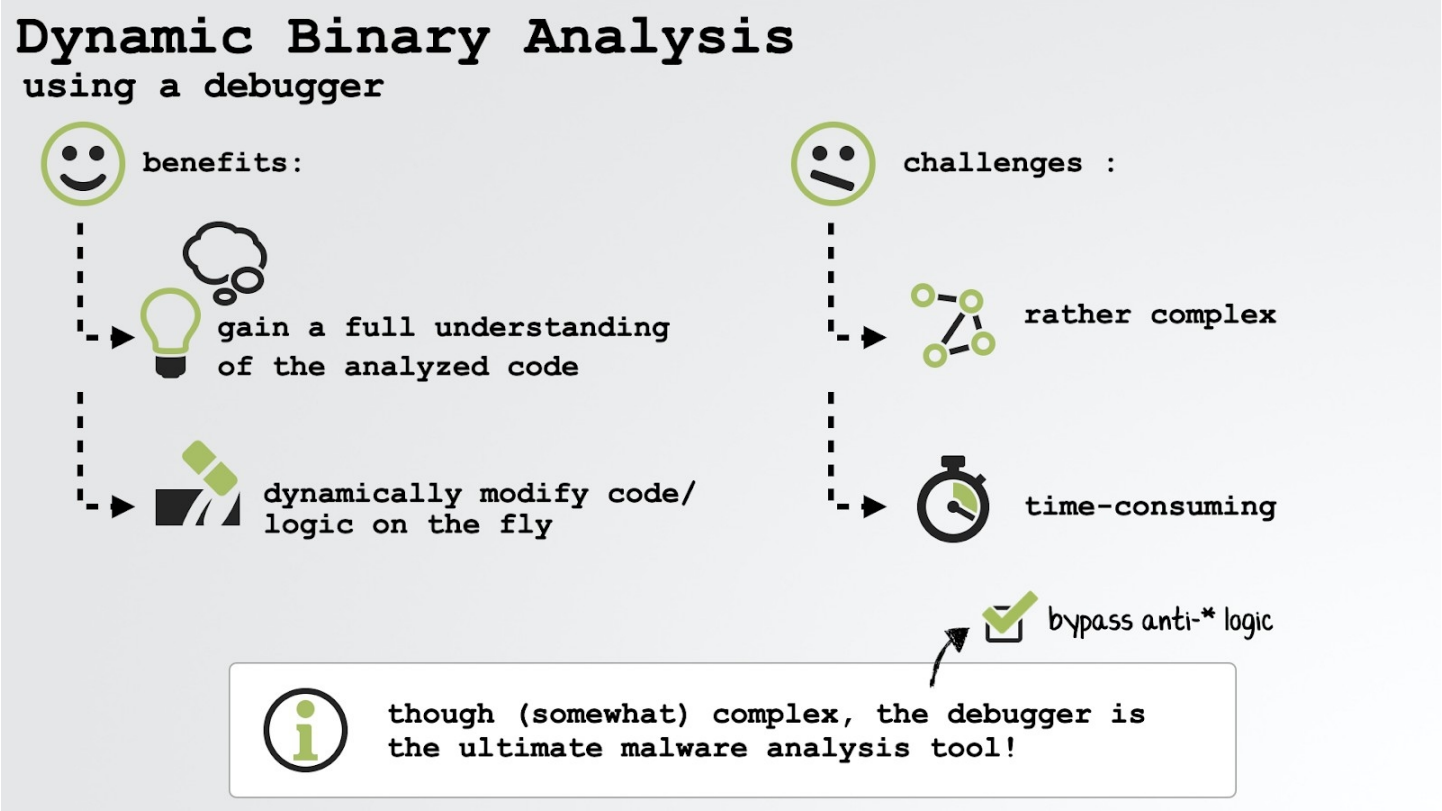
这种分析方法和对嵌入式配置信息的解密最值得注意的一点可能就是几乎不需要动一根手指！相反，通过调试器，我们只允许恶意软件愉快地执行 ...然后（恶意软件不知道），从内存中提取解密数据。

这只是一个例子，说明了调试器的威力！更全面地说，调试器的好处包括：

- 全面理解所分析的代码
- 动态修改代码，例如绕过反分析逻辑

当然，还有一些挑战在一定程度上削弱了这些好处，包括调试器：

- 是一个相当复杂的工具（需要特定的、低级的知识）
- 完成分析可能需要大量时间

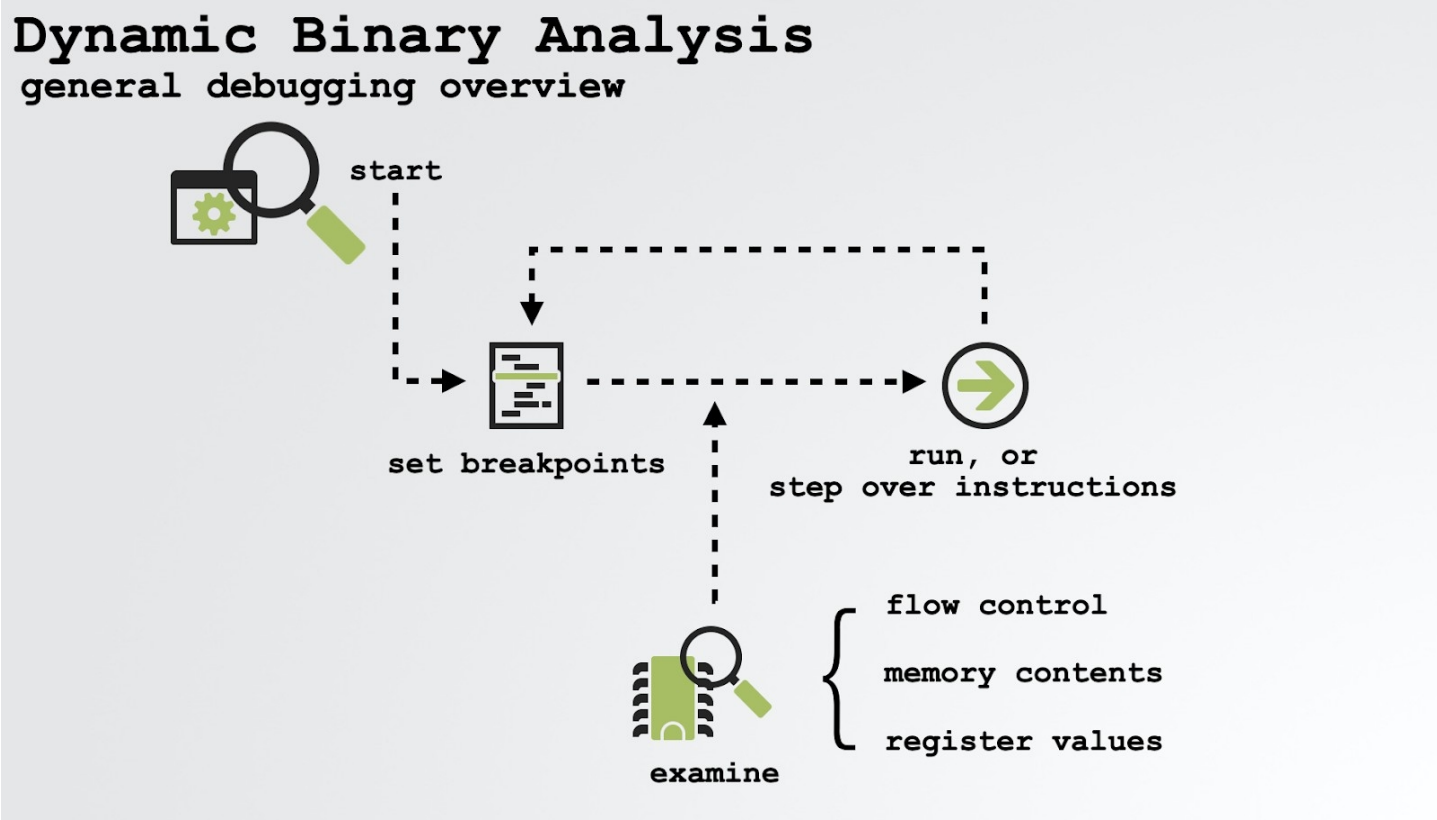


但是，一旦您了解了调试器的概念和技术，能够高效地进行调试，调试器将成为您最好的（恶意软件分析）朋友。

调试概念

在高层，调试会话通常以以下方式流动：

1. 目标进程（即要分析的恶意软件样本）被“加载”到调试器中。
2. 断点设置在代码中的不同位置，例如恶意软件的主入口点或感兴趣的方法调用处。
3. 示例将启动并运行，直到遇到断点，此时执行将停止。
4. 一旦停止，就可以自由地四处查看、检查内存和寄存器值（例如检索未加密的数据）、控制流（即调用堆栈）等等。
5. 然后，执行要么继续（并一直运行，直到命中另一个断点），要么一次执行一条指令。



笔记:

调试恶意样本时，允许其执行（尽管是在检测环境中）。因此，请始终在虚拟机中执行调试。

除了确保不会发生持久性损坏外，虚拟机还可以恢复到以前的状态。这通常在调试会话期间非常有用（例如，当断点丢失且恶意软件全部执行时）。

要启动调试会话，只需将示例加载到调试器中。另一个选项是将其附加到已经运行的进程。然而，在分析恶意样本时，我们通常希望在恶意软件开始执行时开始调试。

笔记:

这里我们重点介绍如何使用lldb,这是macOS二进制调试的实际工具。尽管应用程序（如Hopper）在其上构建了用户友好的界面，但通过其命令行界面直接与lldb交互可以说是最强大、最有效的调试方法。

lldb与Xcode（/usr/bin/lldb）一起安装。但是，如果希望只从终端安装lldb：键入lldb,点击enter,并同意安装提示。

lldb网站[2]提供了丰富的详细知识，例如该工具的深入教程[3]。

此外，任何命令都可以参考lldb help命令，以提供内联信息。例如，下面描述了在断点上操作的命令:

（lldb）帮助断点

lldb官方教程还指出，有一个“apropos”命令，“将在帮助文本中搜索特定单词的所有命令，并为每个匹配的命令转储摘要帮助字符串。” [3]

在lldb中启动调试会话有几种方法。最简单的是从终端执行lldb，以及要分析的二进制文件的路径（加上所述二进制文件的任何附加参数）:

```
$ lldb ~/下载/恶意软件。bin任意args（

lldb）目标创建“malware.bin”
```

```
当前可执行文件设置为“恶意软件”。bin'（x86_64）。
```

如上所示，lldb将显示一条目标创建消息，记录要调试的可执行文件集，并识别其体系结构。尽管调试会话已经创建，但示例的所有指令（例如malware.bin）都尚未执行。

lldb还可以通过-pid <target pid>连接到正在运行的进程的实例。然而，在分析恶意软件的环境中，这种方法并不常用（因为恶意软件已经关闭并运行，因此实际上可以防止这种连接）。

最后，在lldb外壳中，可以利用 --waitfor命令，“它等待具有该名称的下一个进程出现，并附加到它” [3]。

```
$ lldb

（lldb）进程附加 --命名恶意软件。箱子 --等待
```

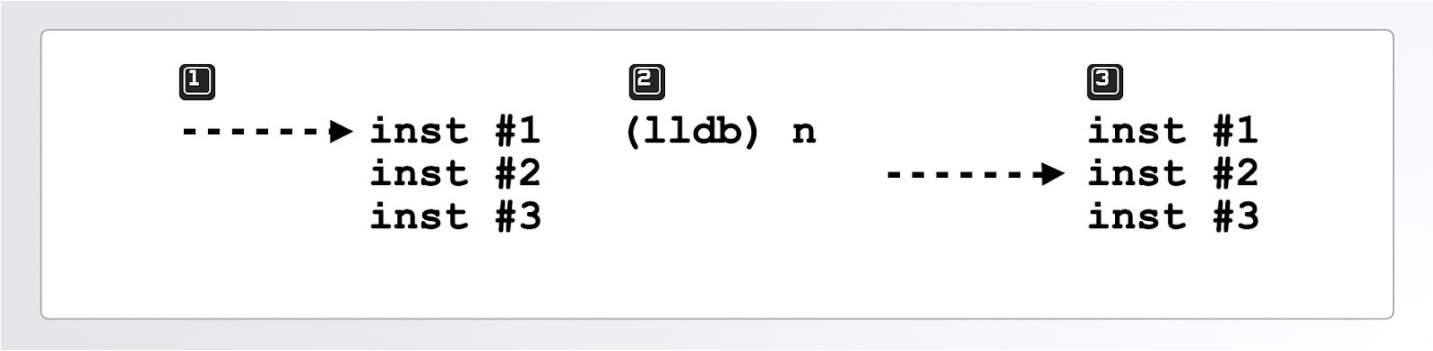
现在，每当一个进程被命名为恶意软件。bin启动后，调试器将自动附加:

```
（lldb）进程附加 --命名恶意软件。箱子 --等待
...

进程14980已停止
*线程#1,队列= 'com。苹果主线程'，停止原因=信号SIGSTOP
...

可执行模块设置为“~/Downloads/malware.bin”。 架构设置
为：x86_64h-apple-macosx-。
```

这个 --waitfor命令在恶意软件衍生出您想要调试的其他（恶意）进程时特别有用。

流量控制

在讨论断点（指示调试器在指定位置停止）之前，让我们先简单地讨论一下执行控制。调试器最强大的一个方面是它能够精确地控制正在调试的进程的执行 ...例如，指示进程执行一条指令（然后停止）。



下表介绍了与执行控制相关的几个lldb命令:

（lldb）命令	说明
跑（右）	运行调试过的进程。 开始执行，直到遇到断点或进程终止为止。
继续（c）	继续执行已调试的进程。 与run命令类似，执行将继续，直到断点或进程终止。
nexti (n)	执行程序计数器（rip）寄存器指向的下一条指令，然后停止。此命令将跳过函数调用。
stepi (s)	执行程序计数器（rip）寄存器指向的下一条指令，然后停止。与nexti命令不同，该命令将进入函数调用。
完成（f）	执行当前函数（“帧”）中的其余指令返回并停止。
控制+ c	暂停执行。如果进程已经运行（r）或继续（c），这将导致进程停止 ...无论它目前在哪里执行。

笔记:

一般来说，lldb与gdb命令保持向后兼容性（gdb是GNU项目调试器，通常在lldb之前使用）。例如，对于单步，lldb既支持“线程步骤inst”，也支持与gdb匹配的“步骤”。

为了简单起见（以及由于作者对gdb的熟悉），我们通常描述与gdb兼容的lldb命令名。

最后，请注意，大多数命令可以缩短为单字母或双字母。例如，“s”将被解释为“step”命令。

有关gdb到lldb命令的详细映射，请参见:

“GDB到LLDB命令映射”[4]

虽然我们可以单步执行二进制文件的每个可执行指令，但这相当乏味。另一方面，简单地指示调试器允许被调试器运行（不受限制），从一开始就违背了调试的目的。“解决方案”？断点！

断点

断点是调试器的“命令”，指示调试器在指定位置停止执行。通常在二进制文件的入口点（或其构造函数之一）、方法调用或相关指令的地址上设置断点。如前所述，一旦调试器停止执行，就可以检查被调试器的当前状态，包括其内存和CPU寄存器内容、调用堆栈等等。

使用breakpoint命令（简称b），可以在指定的位置（例如函数或方法名）或地址设置断点。

假设我们想要调试一个恶意样本（malware.bin），并且想要在其主要功能处停止执行。启动lldb调试会话后，我们可以简单地键入b main:

```
(lldb) b main
断点1：其中=恶意软件。宾曼，
              地址= 0x0000000100004bd9
```

设置此断点后，如果我们随后指示调试器（通过run命令）运行被调试的进程，执行将开始，但在到达主函数（开始）处的指令时停止:

```
（lldb）运行

（lldb）进程1953停止原因=断点1.1
->  0x100004bd9 <+0>: pushq  %rbp
```

很大一部分Mac恶意软件是用Objective-C编写的，这意味着（即使是以编译的形式）它将同时包含类名和方法名。因此，我们还可以在这些方法名上设置断点。怎样只需将完整的方法名传递给breakpoint（b）命令即可。例如，要中断NSDictionary类的objectForKey: method,请键入：b -[NSDictionary objectForKey:]。

> 笔记:
>
> 有关调试Objective-C代码的深入讨论，请参阅优秀的writeup: “Dancing in the Debugger – A Waltz with LLDB” [5]

如前所述，断点也可以按地址设置（例如，用于在函数中设置断点）。要在地址上设置断点，请指定前面带有0x的（十六进制）地址。例如，我们还可以通过：b 0x0000000100004bd9在恶意样本（位于地址0x0000000100004bd9）的主函数上设置断点。

断点可以通过下面描述的命令列出或删除:

（lldb）命令	说明
断点（b）<函数/方法名称>	在指定的函数或方法名上设置断点。
断点（b）<地址>	在指定内存地址的指令上设置断点。
断点列表（br l）	显示（列出）所有当前断点。
断点启用/禁用（br e/dis）	启用或禁用断点（由数字指定）。
断点删除<#>	删除断点（由数字指定）。

help命令（带有断点参数）提供了一个完整的