2016 年 4 月 28 日，星期四

研究聚焦：QBOT 再度抬头

作者：Ben Baker。

Qbot（也称为 Qakbot）估计早在 2008 年时就已出现，但是最近它的开发和部署却大有抬头之势。Qbot 正在转变为主要以银行凭证等敏感信息为目标的 Qbot。我们通过本文揭露该恶意软件尚未为大家所知的最新变化。

Qbot 主要以浏览器漏洞攻击包负载的方式造成感染。网站管理员通常会使用 FTP 来访问服务器，因此 Qbot 会尝试窃取 FTP 凭证，以便将这些服务器添加到其托管恶意软件的基础设施中。Qbot 也可以通过服务器消息块 (SMB) 在网络之间传播，因此，要从未受保护的网络中清除它是非常困难的。

加壳程序

Qbot 使用了一种可变的加壳程序，在不同的样本之间变化十分显著。该加壳程序的字符串和代码块随机变化，所以很难对其创建检测签名。随机化是 Qbot 的常用手段，其使用的文件名、域名和加密密钥都是随机生成的。

幸运的是，代码混淆并没有使加壳程序代码的行为变得复杂。解壳代码总是在内存中建立整个 PE 文件（可能是最初未解壳可执行文件的 MD5 匹配文件），然后使用自定义加载程序执行已解壳的文件。当加载程序调用 Windows API VirtualProtect 时，可以将解壳的可执行文件从内存中可靠地提取出来。因为解壳代码使用 VirtualProtect 来保证已解壳的内存部分得到执行，所以我们有可能在已解壳的代码得到执行并感染 VM 之前，对其进行转储。

为了利用解壳程序的可预测性，我们使用 Pykd 脚本对每个加壳文件进行调试。我们在 VirtualProtect 中设置了一个断点，然后扫描该进程的内存，以便转储已解壳的可执行文件。我们在已解壳代码执行之前就终止其进程，因此，在样本之间无须恢复 VM。此脚本每秒可以解壳多个样本，所以我们能够解壳大量样本。

我们分析了 618 个加壳的样本，所有这些样本解壳为 73 个不同的样本。每个样本都以多种不同方式加壳，以使散列签名的有效性降低。

安装

当加壳程序在内存中加载已解壳的可执行文件后，Qbot 便会检查它是否已完成安装。如果 Qbot 未从“%appdata%\Microsoft\[RandomName]\ [RandomName].exe”运行，则会从该位置执行自我复制，并执行副本。

Qbot 通过向梅森旋转算法随机数生成器植入字符串或 SHA1 散列的 CRC32 校验和，生成随机字符串，然后重复产生随机整数，作为字母阵列的索引，以挑选字母。它会使用从 ProductId（位置为 HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductId）生成的种子、计算机的名称（通过 Windows API GetComputerNameA 获得）和硬盘驱动器序列号（通过 Windows API GetVolumeInformationA 获得）生成安装文件夹的名称。

Qbot 使用确定性的字符串生成器，因此如果样本在同一台计算机上运行，就始终会产生相同的文件名和互斥体。例如，如果沙盒使用静态产品 ID、计算机名称和卷序列号，则文件名看似是静态的，但是在任何其他计算机上都不尽相同。这导致很难为客户创建通用的 IOC，不过，创建沙盒 IOC 来识别 Qbot 还是很容易实现的。

当 Qbot 在其安装目录中得到执行后，它将执行 Windows 可执行文件“Explorer.exe”的一个新实例。注入进程加载含有恶意 DLL 的资源“IDB_BITMAP1”。该资源使用前 20 个字节作为 RC4 密钥来解密。解密后的数据含有压缩形式的 DLL，位于压缩数据的 SHA1 散列之后。

压缩看起来是自定义的，但是与 LZSS（带有指向作为词典的现有解压缩数据的偏移长度对）相似。文件被分解为压缩块，每一个开头都是如下格式的 24 字节报头：

Magic Bytes[8]: "\x61\x6C\xD3\x1A\x00\x00\x00\x01"
Compressed Data Size[4]
Compressed Data CRC32[4]
Decompressed Data Size[4]
Decompressed Data CRC32[4]

DLL 被分解为 3 个压缩块，这意味着从第一次解密开始，Qbot 就会使用 SHA1 散列来检查 6 个 CRC32 校验和。要加载一个 DLL，就要进行大量错误检查。

日志记录

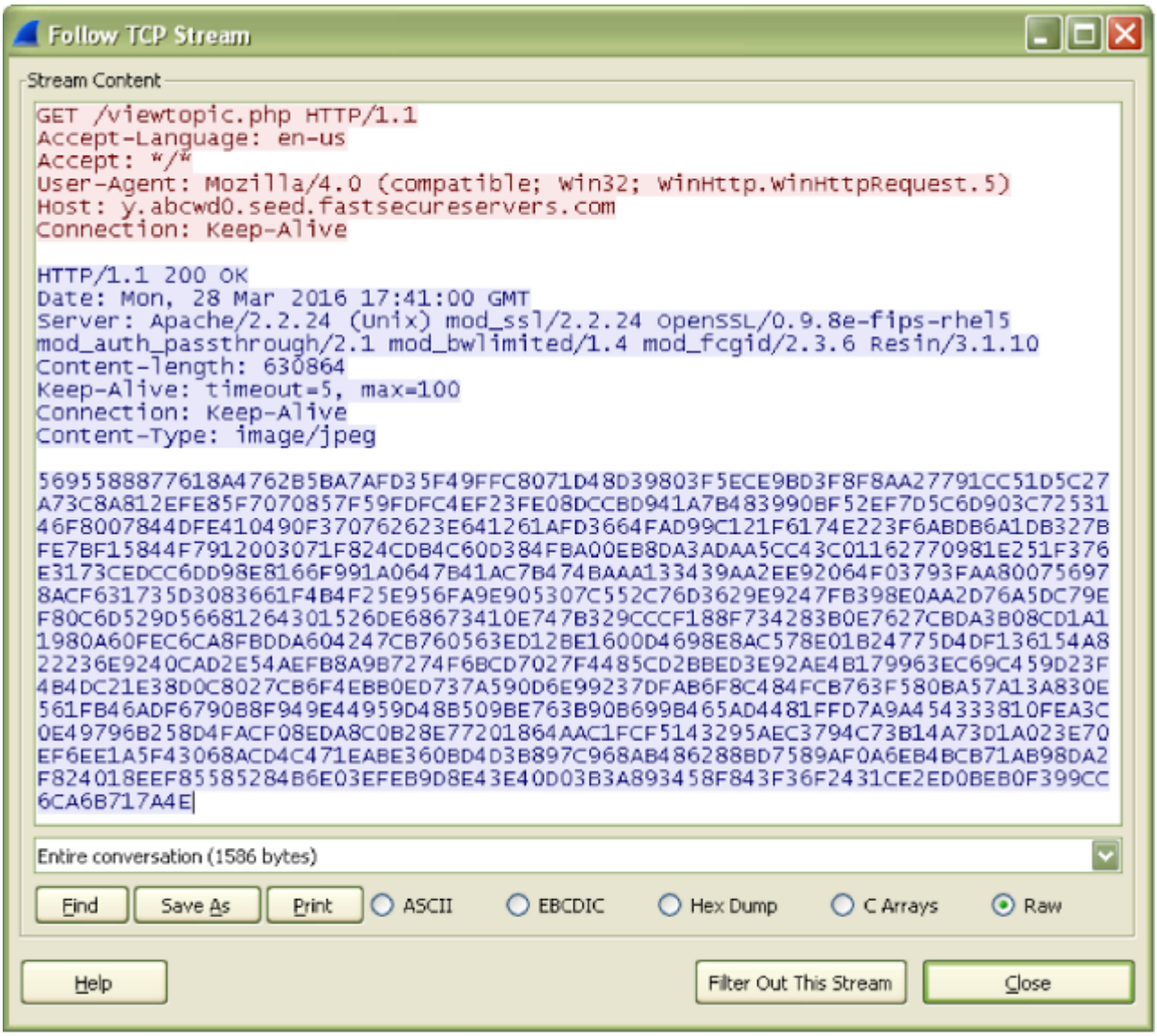
Qbot 将日志记录在安装路径的一个加密文件中。日志文件具有 DLL 扩展名，而文件名则比 Qbot 的安装目录名称少一个字母，所以可以识别出日志文件。例如，如果 Qbot 安装在“%appdata%\Microsoft\Oykyjxjx”中，则日志文件为“%appdata%\Microsoft\oykyjxjx\oykyjxj.dll”。日志文件使用 RC4 密钥加密。RC4 密钥是通过将文件夹名称转换为小写，然后采用所产生字符串的 SHA1 散列。

我们创建了一个简短的 Python 脚本，用于解密日志文件，以便事件响应程序能够获得关于感染的更多信息。这将打印配置信息，包括初始感染时间和 FTP exfil 服务器信息。

```
from Crypto.Cipher import ARC4
from Crypto.Hash import SHA

def Decrypt(data, filename):
    h = SHA.new(filename)  # SHA1 Hash filename for RC4
    key
    key = h.digest()
    cipher= ARC4.new(key)
    return cipher.decrypt(data)

filename = "oykyjxj.dll"
fullpath = "C:\\Qbot\\" + filename
data = file(fullpath, "rb").read()

fileroot = filename.split(".")[0]

for i in xrange(0,26): # Brute force the missing letter
    letter = chr(0x61 + i)
    key = fileroot + letter
    decrData = Decrypt(data,key)

    if decrData[0:20] in SHA.new(decrData[20:]).digest():
     # We found the key if the first 20 bytes match the
    SHA1 of the remaining bytes
        print key + ": \n" + decrData[20:]
```

更新程序

Qbot 使用扩展名为“.wpl”的模糊脚本进行自我更新。此脚本尝试从类似于“http[:]//<maliciousdomain.com>/viewtopic.php”这样的 URI 中大量的域下载托管的加密可执行文件。此脚本以十六进制对服务器响应进行解码，然后使用前 20 个字节作为 RC4 密钥来解密其余字节。解密的缓冲区包含 20 字节的 SHA1 散列，后面跟着 Qbot 可执行文件的更新版本。



信息盗窃

Qbot 主要以盗窃银行凭证等敏感信息为目标。它通过窃取存储的 cookie 或凭证等数据，以及通过向网络浏览器注入代码来操纵进行中的浏览会话，从而达到这个目的。Qbot 让恶意攻击者能够利用受害者的浏览会话，使他们能绕过双因素身份验证之类的简单安全措施。

Qbot 最近向其信息盗窃技术系列中添加了 Webinjects。Webinjects，通常与 Zeus 和 Spyeye 相关联，使得在浏览器会话中注入恶意 JavaScript 变得容易，同时还可以记录或者重定向受害者的活动。Webinjects 可以产生强大作用，在某些情况下能够自动进行大量银行业务交易，而无需用户的任何交互。

Qbot 含有可以解析高级 Webinjects（例如自动转账）的代码，但是，我们分析的全部 618 个 Qbot 样本只是在受害者尝试注销在线银行页面时，将浏览器重定向。如果受害者无法注销目标站点，则被盗窃的 Cookie 和会话令牌将很长时间保持活动状态。

示例 Qbot Webinject 配置：
“set_url https://*.<BankDomain>.com/*logoff* GPR
http://<MaliciousSite>/fakes/onlineserv_cm_logoff.html”

在这个 Webinject 示例中，任何被入侵的浏览器在尝试导航到目标银行的注销页面时，都会被拦截。“GPR”标志表示 GET 和 POST 会被拦截，并重定向到恶意 URL。支持 Webinjects 的其他恶意软件似乎很少使用这个 R 标志。关于 Webinjects 的恶意软件论坛帖文主要侧重于注入恶意代码，或者只是简单记录 HTTPS 参数（例如凭证）。

命令和控制

FTP EXFIL

Qbot 通过 FTP 将数据偷偷传输到自己的配置文件中的硬编码列表中的服务器。exfil 文件经过压缩，然后使用随机生成的密钥进行 RC4 加密，类似于资源在可执行文件中的加密。我故意忽略如何解密这些文件，因为披露该信息会使任何人都能够获得 Qbot 盗窃的敏感信息。

这些 exfil 文件会被上传到 FTP 服务器，文件名类似于“article_covezh618946_1450458170.zip”，其中“article”为硬编码，“covezh618946”为随机生成的，而 1450458170 是从 Linux 时间（在本例中，是 2015 年 12 月 18 日）开始后经过的秒数。

解密的 exfil 文件含有关于受害者的计算机上的大量信息，其中一大块是通过使用以下字符串格式调用函数 wvnsprintf 而形成的：

“ext_ip=[%s] dnsname=[%s] hostname=[%s] user=[%s] domain=[%s] is_admin=[%s] os=[%s] qbot_version=[%s] install_time= %s] exe=[%s]”

qbot_version 字符串是通过之前使用格式字符串“%04x.%u”调用 wvnsprintf 而生成的，使用的参数可回溯到数据段的前 2 个 DWORD（双字）。直接提取这些 DWORD 已被证明是一个简单有效的方法，无须运行 Qbot 即可提取该恶意软件的版本信息。

HTTP DGA

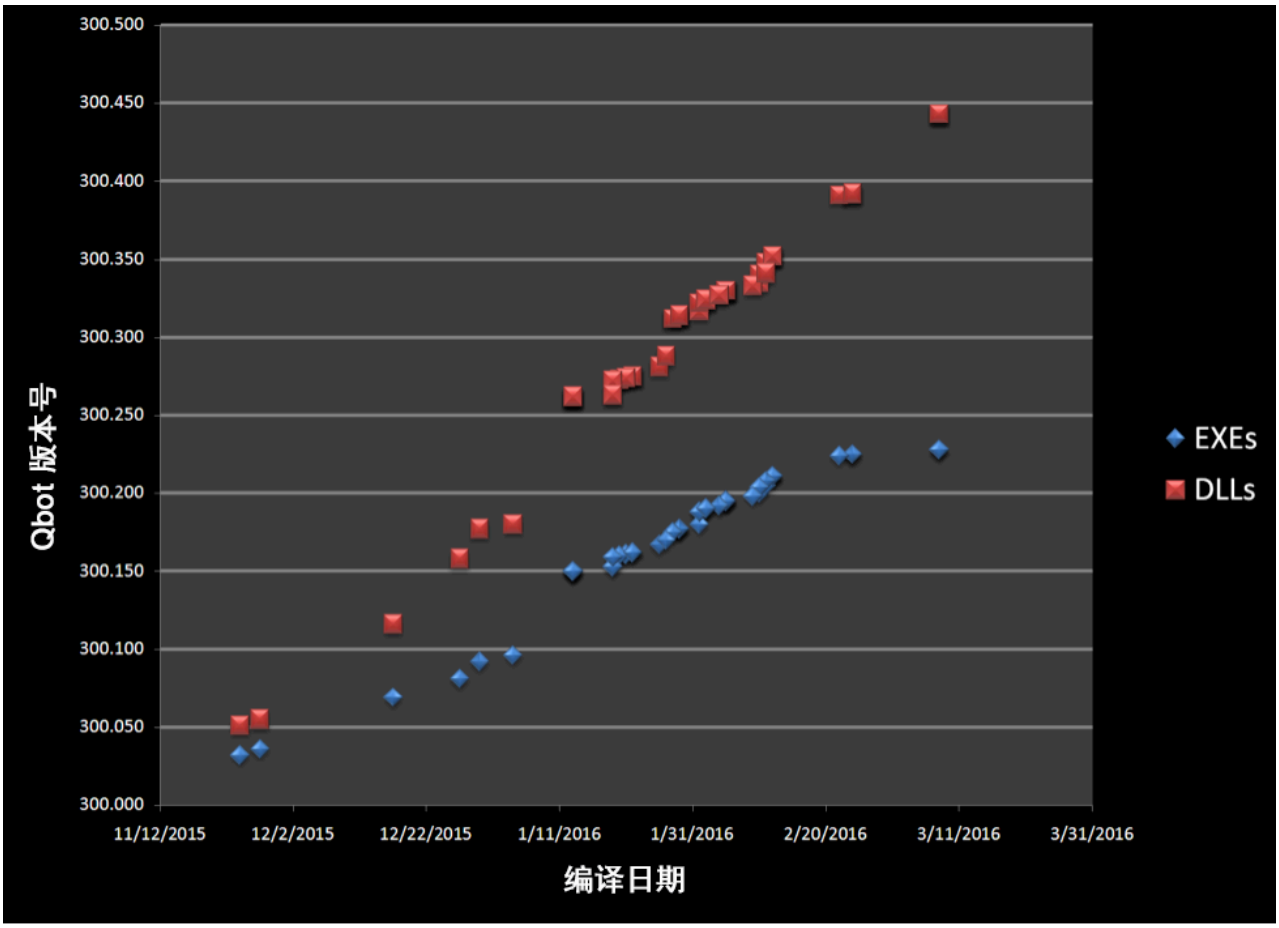
Qbot 的 DGA 生成类似于“2.mar.2016.00000001”的字符串，其中 00000001 为常数，而第一位数是日期的十位数（虽然对月份的 30 日和 31 日也使用数字 2）。这意味着每个月只有 3 个 DGA 种子。Qbot 通过向 Google 发送一个看起来无害的 HTTP 请求，然后解析来自 HTTP 301 响应的日期，来获得日期。

日期字符串经过 CRC32 校验和的校验，并用作梅森旋转算法随机数生成器的种子，而 Qbot 使用它来构建可预测的域列表。Qbot 每次生成一个含有 5 个域的列表，并随机从中挑选，直至找到活动的域。如果第一组的域都不处于活动状态，则会生成新的一组，并根据需要重复。

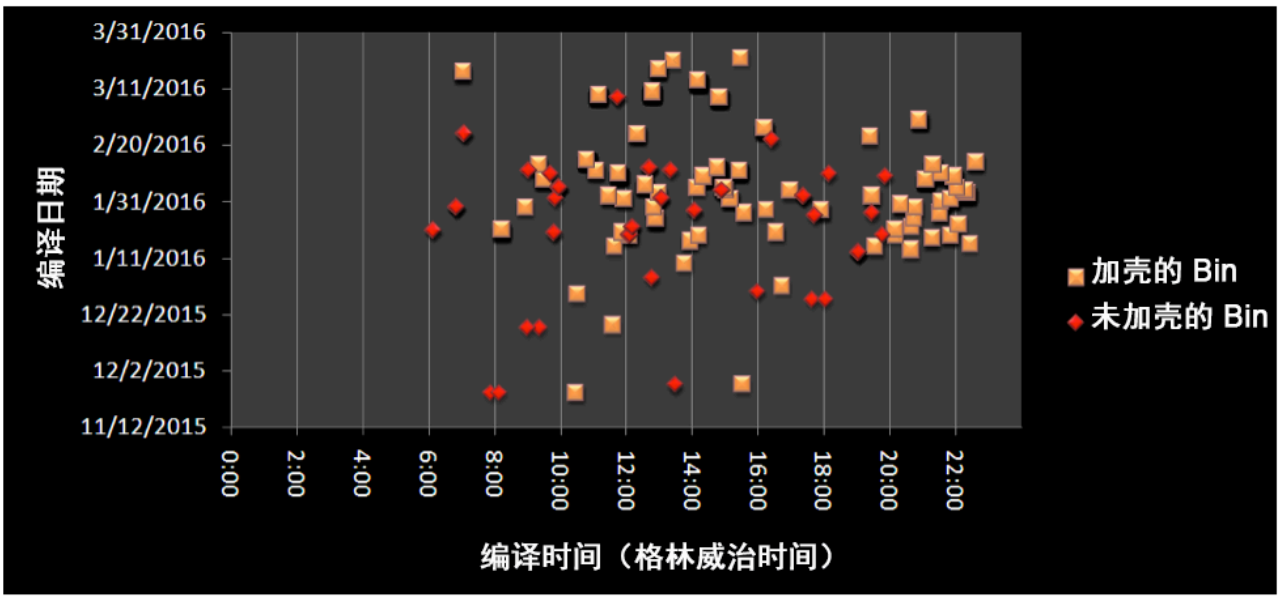
Qbot 检查任何含有“sinkhole”或“.csof.net”的字符串的 DNS 响应，并跳过这些结果。它还检查是否有 Wireshark 这样的监控工具，如果发现有这类工具，则修改种子。修改后的种子会导致恶意软件产生虚假的域列表。

演进

我们通过 Pykd 自动解壳了 618 个 Qbot 样本，然后创建了 Python 脚本来对内嵌的资源进行解密和解压缩。我们提取了每一个文件的 DLL 和配置数据，以及 Qbot 版本信息和编译时间。编译时间经常被用于识别恶意活动，但是，必须注意，编译时间是可以人为修改的。



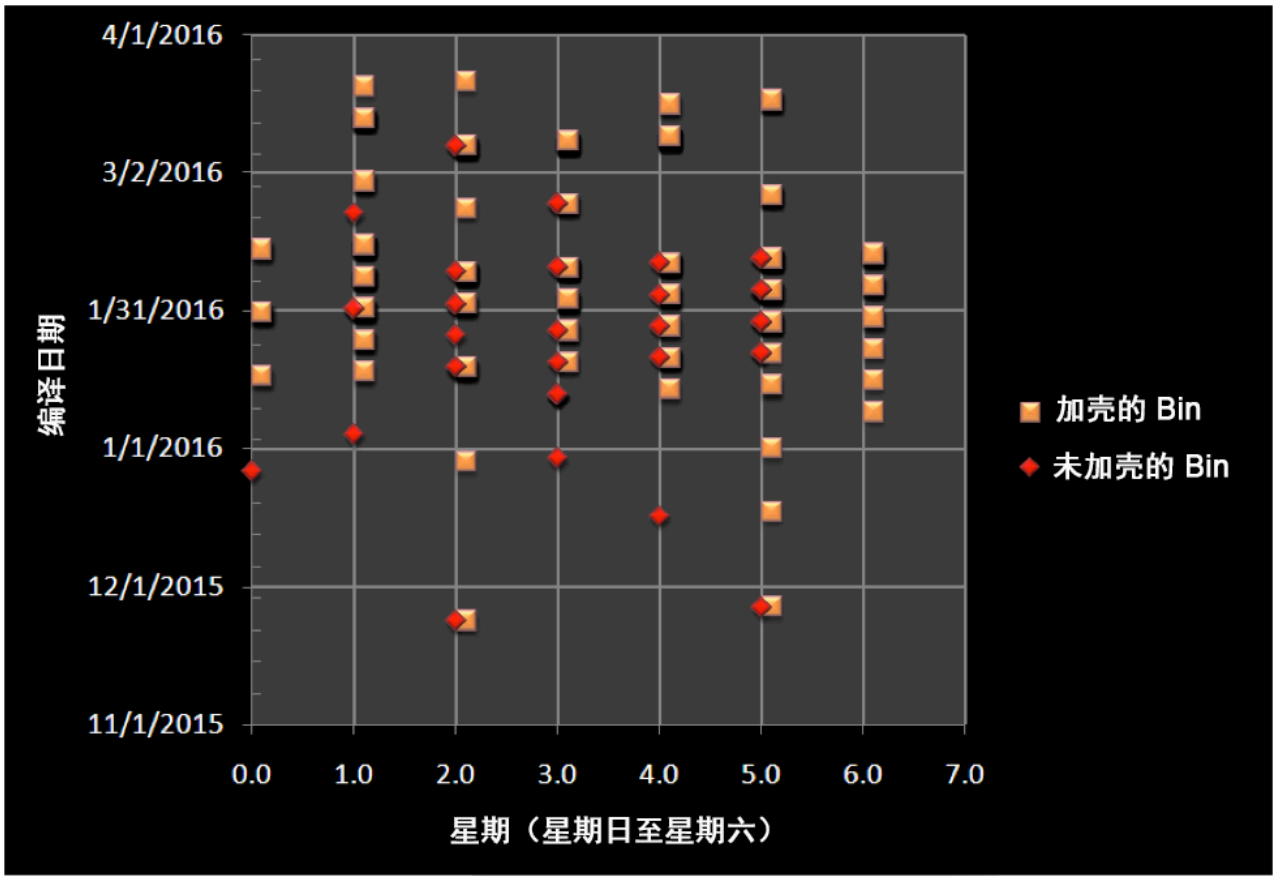
从 Qbot 编译时间可以看出，一月份和二月份的活动大幅增加。在这段时间的至少 4 天内，开发者每天发布了多个版本。Qbot 的制作者似乎在 2016 年 3 月 8 日停止了开发，不过他们的服务器继续托管 Qbot 可执行文件（加壳时间直至 2016 年 3 月 25 日）。



该图展示了 132 个唯一的编译时间戳，显示了每个二进制文件的日期时间。解壳的二进制文件的编译时间都介于格林威治标准时间早上 6 点和晚上 8 点之间。这个时间分布表明攻击者看起来更像是一项全职工作，而不是副业。解壳的二进制文件的编译时间有更多变化，意味着加壳是由另一个时间安排灵活很多的人完成的，或者是利用某种自动化手段协助完成的。开发源代码需要的经验比运行加壳工具要多很多，所以可以由技术低一些的团队成员来完成加壳。

加壳编译时间多数在格林威治标准时间早上 8 点到晚上 10 点，与已解壳样本的平均工作日有 2 小时偏差。如果加壳过程完全自动化，我们应该会发现这些编译时间的分布更随机（因为一些二进制文件的编译时间为午夜到早上 6 之间），或者是通过一个可预测的时间表（例如在每天的某个特定时间执行一次）来完成的。

55% 的加壳二进制文件的编译时间戳在其对应的未加壳文件之后的 2 小时 3 分钟到 2 小时 6 分钟内。这个时间偏差的一致性可能意味着这些文件是根据一个只需要 2 小时多一点的非常一致的流程进行加壳的，或者进行加壳的计算机的系统时钟时间可能被关闭了 2 小时。



该图使用相对于星期日的偏移来表示星期几（例如，1 = 星期一，2 = 星期二，以此类推）。未加壳的二进制文件的编译时间形成了一个完美的星期一到星期五的工作周。在这个为期 17 周的时间段内，只有一组二进制文件是在周末编译的。

加壳的二进制文件继续显示出很大的编译时间差异，编译于连续的 6 个星期六。从 1 月 27 日到 2 月 6 日，每天至少编译了一个加壳的二进制文件，说明连续工作了 11 天。

我们还分析了与 Qbot 二进制文件相关的 Rich Header（富标头）。Rich Header 是 Visual Studio 可执行文件中没有记录的部分，含有用于构建可执行文件的编译器和链接器的版本信息。两个看起来相同的开发环境的 Rich Header 可能差异很大，所以它们可能可以提供线索，说明在一个开发项目中使用了多少台计算机。

154 个未加壳的二进制文件仅含有 6 个唯一 Rich Header，其中有些几乎一样，可能是由同一台计算机上细微的编译器更新产生的。这些标头显示，未加壳的二进制文件是在 3 个不同的环境（可能是不同的计算机或虚拟机）中编译的。加壳的二进制文件含有 44 个唯一 Rich Header，其中 35 个似乎是其余的细微变体。加壳的二进制文件看起来是在 9 个唯一环境中进行编译的，没有一个和未加壳二进制文件的 3 个 Rich Header 相符。

这些 Rich Header 的差异支持了这个推测：加壳是在与开发和编译主要 Qbot 源代码的计算机不同的其他计算机上进行的。Rich Header 数据表明，有 12 个不同的环境被用于二进制文件的编译和加壳，这可能为我们推测 Qbot 的开发和加壳团队的规模提供线索。

开发和维护恶意软件和恶意基础设施需要大量时间和精力。在大规模犯罪软件（例如 Qbot）后的恶意攻击者会组成团队，并将其不法活动作为全职工作。

发布者：EARL CARTER；发布时间： 下午 5:05