Operation(Đường chín đoạn) typhoon：觊觎南海九段线的赛博海莲

By 红雨滴团队

Archived: 2026-04-05 16:30:47 UTC

概述

去年，奇安信威胁情报中心发布了《Operation（Thủy Tinh）OceanStorm：隐匿在深海巨渊下的邪恶莲花》[1]。此后海莲花的攻击活动不降反增，并在2022年入侵某地区的IOT设备作为跳板攻击对另一个地区的目标进行攻击，并首次发现了海莲花在Arm和mips平台的木马，我们将其命名为“caja”，其中mips的木马是用来针对将要铺开的定制化系统。

本文主要会介绍海莲花在过去的一段时间内网渗透过程中所用的一些手法，并披露其在2021年所使用的三个0day漏洞和数个Nday漏洞，恶意代码方面我们会披露近期使用的一些免杀loader、释放的一些功能模块和基于不同语言的隧道转发工具。

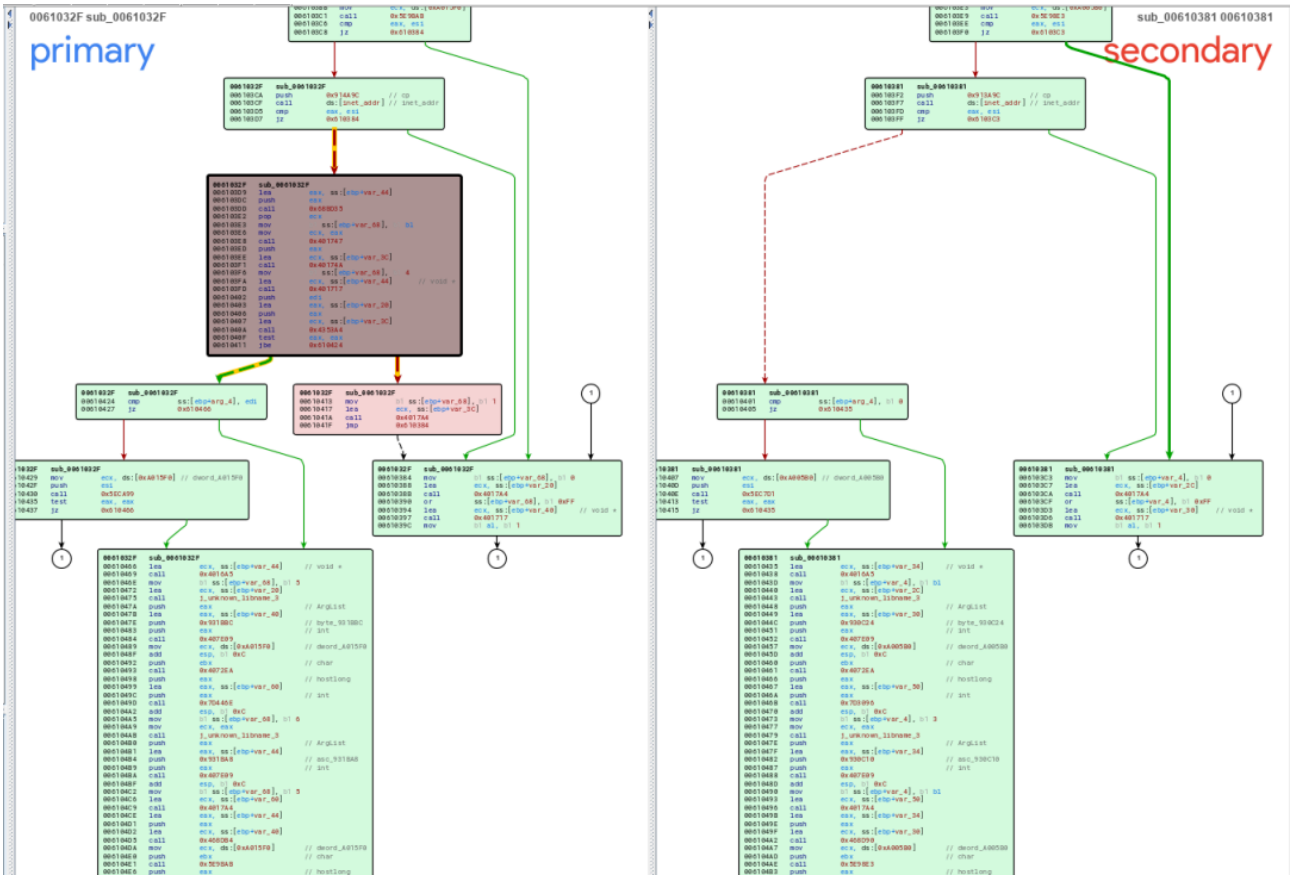
与之前的文章类似，本文内容也仅仅是对海莲花在过去一段时间内攻击手法做一个分享（0day漏洞所涉及的产品均已修补）。供友商追踪溯源。

漏洞分析

0day漏洞分析

0day漏洞一

漏洞出现在某终端管理软件中，服务端和客户端均本地监听了一个2开头的高端口（2XXXX），与该端口通信数据走的是私有协议，漏洞出现在对发来的数据包进行判断的逻辑。



这部分代码由于开发人员的疏忽在strfind之后没有将返回值置为-1。

导致在代码后续执行过程中会执行从数据包中传来的命令

海莲花利用此漏洞可以在内网中进行漫游，对装有某终端管理软件的机器执行任意命令。

0day漏洞二

漏洞出现在某终端管理系统服务端web管理页面，由于登录页面需要令牌验证，安全性很高，但海莲花可能在之前拿到了整站源码找到一个隐藏的登录页面（该页面只需要输入账号密码没有多余的验证），并且拿到了登录的账号密码，成功登录到web端管理后台，并通过其中的一个带有漏洞页面，写入恶意脚本，从而实现命令执行。

0day漏洞三

某杀毒软件服务端存在RCE漏洞，海莲花使用该漏洞入侵杀软服务器端，并向服务端下的所有机器下发木马。

渗透以及横向移动

海莲花非常善于使用测绘平台对全网进行测绘。在使用Nday漏洞时并没有我们想象的具有定向性，批量入侵所有带有漏洞的站点并进行筛选，如果存在高价值的目标则进一步横向移动，其余的当作代理方便海莲花的活动。例如，我们曾在一个色情网站上发现了海莲花投递的webshell和木马。

海莲花利用老版本showdoc任意文件上传漏洞上传webshell

接着上传了使用开源loader生成的CS木马。

木马回连的C2为海莲花控制的IOT跳板，海莲花从去年初就开始对路由器、摄像头等IOT设备进行攻击，入侵成功后植入tinyPortMapper端口转发软件，将跳板服务器的高端口的流量转发到攻击者真正的CS服务器上。

之后的一段时间，海莲花开始对韩国地区的一些三星路由设备进行攻击并当作跳板，但整体数量不是很大，我们推测海莲花可能在韩国也有相应的“政治任务”，而到了今年上半年开始针对台湾地区的IOT设备进行攻击，与IOT设备建立连接后，海莲花会尝试在IOT设备上上传busybox或者dropbear便于攻击者进一步渗透。

上传的Busybox如下所示

上传的dropbear如下图所示

最终攻击者植入的了一个基于arm架构的linux木马，方便长期控制，之后会根据业务方向设置转发，最近我们观察到海莲花将转发软件从tinyPortMapper换成了Gost。

在内网横向移动方面，我们观察到海莲花使用CVE-2020–14882 WebLogic远程命令执行漏洞、CVE-2021-22986 F5 BIG-IP iControl REST未授权远程命令执行漏洞和Jboss反序列化漏洞在内网中漫游，然后会在内网重要节点的服务器上创建Powershell Webserver，并在上面存储加密的CS payload。

代码源于github项目Powershell-Webserver

通过powershell内存加载mimikatz的方式拿到域管的账号密码后开始使用windows.vbs进行横向移动投递木马。

经过分析发现该VBS源于github项目WMIHACKER

投递的木马会向内网powershell服务器请求加密的payload并加载。

最近我们观察到海莲花针对运维人员的攻击活动，通过CNVD-2022-03672漏洞拿下运维人员的电脑，从桌面的密码本上窃取了连接服务器todesk服务的口令，之后海莲花通过todesk远程登录服务器手动拷贝带有木马文件的压缩包，并右键解压执行。

木马载荷介绍

Windows

最近我们观察到海莲花开始使用亚马逊的cdn云函数作为CobaltStrike的C2服务器。

Loader

海莲花会使用各式各样的免杀loader去加载CS后门从而绕过杀软查杀，如使用NIM语音编写的loader

或者使用github上最新开源的loader，

Md5	**文件名**
b217729d1da6363232d4724f9696b6bb	SecurityHealthService.exe
3c7c1dc5f88267fc2cec0b8e412a5a49	LOG.Notify.Implementation.dll

这两个Loader分别为shhhloader和Mortar Loader

除此之外还会使用“msbuild.exe 源码”的方式加载CS

Md5	命令行
64e7ec81190ecada6848dae767c740eb	Msbuild.exe mshtml5.tlb

源码内容如下：

隧道木马

除了CS后门，海莲花还会使用开源或者魔改的隧道木马来与目标服务器建立连接

Md5	文件名
0152aadab26b41e3f53c95096d0e3a2e	DownloadSDK.exe
c93ceb5c5983e1c2ddfa035ee398022c	

第一个样本为python打包的exe，解出来脚本如下：

经过对比发现为开源隧道工具RPIVOT

第二个样本为Golang编写隧道木马，该样本一般会注入到系统进程中不落地， 我们推测海莲花是基于smux和cli两个库自行开发的隧道木马。

插件

目前我们仅从内存和日志中捕获到以下三个插件

Md5	功能
7913d1a143e6fd3a8621fd22bb712200	PS脚本用于窃取浏览器信息并上传到图床

69c7a7c5adda09c3ce6e47c7e76d7e7e	挂起系统日志进程
4B2DF0098372485369E8088EEB4CD554	添加用户并禁用UAC远程限制

窃密PS脚本如下：

接着将获取到的数据通过内网代理的方式加密发送到图床网站上。

海莲花只需要下载图床对应的URL即可拿到相关数据。

插件二的功能逻辑如下：

找到日志服务的进程PID，接着挂起该进程。

插件三的功能逻辑如下，加载资源节中的shellcode

添加新账号，账号名称和密码分别为 "DefaultAccount2"、"!2345^Abc"

把新账号加入管理员组，并禁用UAC远程限制

Linux

Linux平台我们一共发现了三种架构的木马，分别为x86、Arm、Mips。通过对样本进行分析我们发现三种架构的木马是基于一套源码交叉编译而来，其通讯协议完全相同，我们将其命名为“caja”木马。我们以Mips架构的为例做详细分析

Md5	**架构**
641b7ab0d42d283f5b34de84cdcdcdc6	Arm

a2d70e7ab7dccf5efcc32b5bbfdecad9	Mips
a54330bc0fdc9c9585f6024dde340177	X86

caja 善用 XOR 编码和 AES 加密的方式隐藏关键信息。其用到的关键字符串，全部用硬编码的固定 Key 进行 XOR 编码处理，在运行中动态解码后再使用。而网络流量中的信息，则结合了 AES 加密和随机 XOR 编码两种方式。值得一提的是其上线包中包含的多个字段，每一个字段的 XOR Key 都不相同，且仅会对目标数据中处于奇数序 的字符做 XOR 编码，对偶数序的字符进行逻辑非处理。caja 可以接收 C&C 下发的 13 种指令，恶意功能主要有以下几种：

- 释放文件、执行文件
- 访问指定 URL 下载文件
- 检查指定文件
- 读取指定文件中指定偏移、指定长度的数据
- 删除指定文件/目录
- 执行 Shell 命令

caja 的整体执行流程如下：

caja 与双头龙 和 Buni 功能不同，不是同一个家族的变种。其中 Buni 由 C++ 编写，caja 与双头龙都由 C 编写。不过它们之间还是有一些类似之处：

- 都惯用 AES 加密和 XOR 编码
- 对于失陷主机的 root 用户和非 root 用户区别对待
- 网络数据包整体结构，都是 Header, Key, Payload 三段拼接
- 都会用 cat /etc/*release ，cat /etc/issue 等命令来获取系统信息
- AES 加解密的 IV 都是硬编码的 \x00\x00\x00\x00\x00\x00\x00\x00

当然，不同之处也很明显：

- 双头龙的关键字符串会用 Stack String 的方式做隐藏，caja 没有使用 Stack Strings
- caja 的 C&C 由硬编码 Key 的 XOR 编码处理，而双头龙的 C&C 则使用 AES 加密和 Rotate 编码
- 双头龙的 AES S_Box 是动态生成的，而 caja 的 AES S_Box 是硬编码的
- 双头龙还会对关键信息进行 ZLib 压缩， caja 没有使用压缩处理

- 创建守护进程的方式不同，双头龙会直接调用 daemon() 系统接口来创建守护进程，caja 会有条件地通过手动 fork 的方式创建守护进程
- 指令码没有相同之处，各个指令对应的功能也大不相同

caja 进程启动时，如果传入参数的数量（argc) **小于或等于 2** ，则会创建守护进程，否则就以普通进程的方式运行：

caja 的 C&C 通过固定的 Key 0xFFABFACB 经过 XOR 循环解码得来：

caja 样本中还有很多关键字符串，都用与 C&C 编码同样的方式做了循环 XOR 编码，解码之后如下：

-
-
-
-
-
-
-

```
/dev/urandom  /dev/random  /var/lib/libuuid/clock.txt  clock: %04x tv: %lu %lu adj: %d\n  clock: %04x tv: %016
```

其中以下几条命令用来收集失陷主机信息：

-
-
-
-
-

```
cat /proc/cpuinfo 2>/dev/null  cat /proc/meminfo 2>/dev/null  cat /proc/version 2>/dev/null  cat /proc/partiti
```

以下 4 个目录，caja 会探测是否有写权限后，选择一个目录写入单一实例的 Lock 文件：

-
-
-
-
-
-

```
# 针对 root 用户的单一实例 Lock 文件目录  /dev  /etc    # 针对非 root 用户的单一实例 Lock 文件目录  /var/tmp
```

caja 会获取失陷主机本地大量的系统信息、网络配置信息和用户信息，其中网络接口名和网卡硬件地址会被拼接起来计算一个哈希值，并把长度为 16 字节的原始二进制哈希值保存到本地文件中。这一串哈希值，可以看作是 caja 为每一台失陷主机生成的 UUID。

该文件的文件名生成逻辑与上述单例 Lock 文件类似，不过种子字符串则是另外两条：

-
-
-
-

```
# 非 root 用户  TfpnZdTBdWnUyiYQPlveDFywVbvWHOfATupSZiVEGLyfTYVJkSNpxNLUrIRYwyySSqmtyiAfTYMoXjXJbnYaCJwTUPRqnS
```

生成的目标文件名：

- root 用户：.MgjwOABSFaQ
- 非 root 用户：.HOfATupSZiV

文件最终存入的目录为以下目录之一：

- /dev/
- $HOME
- /etc/
- 当前进程文件所在的目录
- /var/tmp/
- /tmp/

执行起来后会随机伪造一个进程名

1. 生成 10-15 个随机小写字母字符串
2. 将上述字符串用 [] 括起来，形成一个如 [lkjsdiorethgn] 的进程名

接着获取本机信息，之后进入生成上线包的流程，caja 上线包的 Payload 部分，按顺序包含以下信息：

- UUID Bytes
- Magic Number(Version Number)
- hostname
- pid
- 未修改进程名之前的进程文件路径
- 网卡接口及 IP 地址
- 系统和网络配置信息
- 一系列 cat 命令执行查到的详细系统配置信息

第二个字段是硬编码在样本中的一个常量数值：0x32 ，这个数值在 caja 的 MIPS 版和 x86-64 版本的样本中是相同的，所以我们可以把它当作是一个版本号，或者标志此款木马的 Magic Number。

其中涉及的三个主要函数 enc_and_append_field_data_*()，主要功能是对 int 类型的数据（主要是 PID）、c_string 类型的数据（各种系统信息）和目标数据对象分别做的编码和拼接处理。对于 int 类型数据，直接以 4 字节的 int_32 类型表述，对于 c_string 类型的数据，先转成目标数据对象在处理。目标数据对象结构如下：

-
-
-
-

```
struct payload_field {       char* data_ptr;       int data_len;   }
```

以其中 UUID 字段的处理流程为例：

分为以下几个步骤：

1. 生成随机的单字节 XOR Key
2. 对 UUID 字节串中的每个字节，奇数序的字节做 XOR 编码，偶数序的字节做逻辑非运算
3. 将 UUID 的 data_len/XOR Key/编码后的字节数据拼接起来，形成如下结构：

后面的每个 Payload 字段，都用相同的逻辑进行编码、拼接处理，最后整体的结构如下：

可以看到最前面的 uuid 字段，前 4 个字节表示长度 0x10，第 5 个字节 0x3d 是本字段的随机 XOR Key，后面 0x10 的字节就是实际的数据字节，后面依此类推。

生成Payload完成后开始进入Header包构造流程在样本中用长度为 0x35 的结构体表述，结构体对象创建代码如下：

上线包 Header 结构体创建之后，会对以下关键数据做初始化：

- 生成 [0x5, 0xF] 之间随机长度的一段数据，作为 Payload AES 加密的密钥。该密钥的长度存放在 Register Header 偏移 0x08 处，密钥数据的地址存放在偏移 0x25 处；
- 把前面生成的 Register Payload 数据的地址赋予 Header 偏移 0x2D，数据长度存放于 Header 的偏移 0x0B 和 0x15 处
- 在 Header 偏移 0x11 处设置上线包的指令码为 0xB76E

前面随机生成的长度为 [0x5, 0xF] 之间的 XOR Key，此处将前序环节编码、拼接好的上线包 Payload 再做一次 AES-256 CBC 模式的加密，AES 的 IV 为硬编码的 \x00\x00\x00\x00\x00\x00\x00\x00。

前面处理好了上线包的 Payload，caja 就把上线包结构体对象的前 0x25 个字节，和上线包 Payload 用到的 AES 密钥以及加密过后的上线包 Payload 拼接起来。步骤如下：

最后，caja 为上线包的 Header 也做了单独的编码，编码过程与前面 Payload 各字段的处理稍有不同：

1. 从第 6 个字节开始，先进行逐字节编码（XOR/逻辑非和其他），XOR Key 为前 5 个字节
2. 再把第一步编码过后的 0x25 个字节做翻转(Reverse)

最终形成的上线包概要结构如下：

指令功能部分如下：

指令码	**功能**
0xB76E	设置 C&C 会话 timeout 为 180s，如果超时，Bot 要主动断开并重新连接。此功能具有一定的反调试能力
0xF76F	重新连接 C&C
0xBB32	释放文件到 $HOME 或者当前进程文件所在目录
0xE04B	检查指定文件的 stat
0xEBF0	终止当前木马进程
0xF28C	从给定的偏移处，读取指定文件的给定长度的数据
0xC221	访问给定 URL，下载文件
0xDEB7	删除给定的文件或目录

0xA16D	Sleep 给定的秒数
0xA863	释放并且执行文件
0xAE35	执行给定的 Shell 命令
0x5B77/0x73BF	心跳
0xA09C	重置 Bot 的 recv buff

对于 C&C 响应数据包的结构，与上线包结构一致，也分 Header, AES_Key, AES_Encrypted_Payload 三段，解密/解码逻辑为前面上线包数据加密/编码过程的逆序即可。

总结

目前，基于奇安信威胁情报中心的威胁情报数据的全线产品，包括奇安信威胁情报平台（TIP）、奇安信天狗漏洞攻击防护系统、天擎、天眼高级威胁检测系统、奇安信NGSOC、奇安信态势感知等，都已经支持对此类攻击的精确检测。

IOC

MD5：

b217729d1da6363232d4724f9696b6bb

3c7c1dc5f88267fc2cec0b8e412a5a49

64e7ec81190ecada6848dae767c740eb

0152aadab26b41e3f53c95096d0e3a2e

c93ceb5c5983e1c2ddfa035ee398022c

7913d1a143e6fd3a8621fd22bb712200

69c7a7c5adda09c3ce6e47c7e76d7e7e

4B2DF0098372485369E8088EEB4CD554

641b7ab0d42d283f5b34de84cdcdcdc6

a2d70e7ab7dccf5efcc32b5bbfdecad9

a54330bc0fdc9c9585f6024dde340177

参考链接

【1】https://mp.weixin.qq.com/s/dGW0FrbZZ5UA6KuuZB8J_g

点击阅读原文至**ALPHA 5.0**

即刻助力威胁研判
