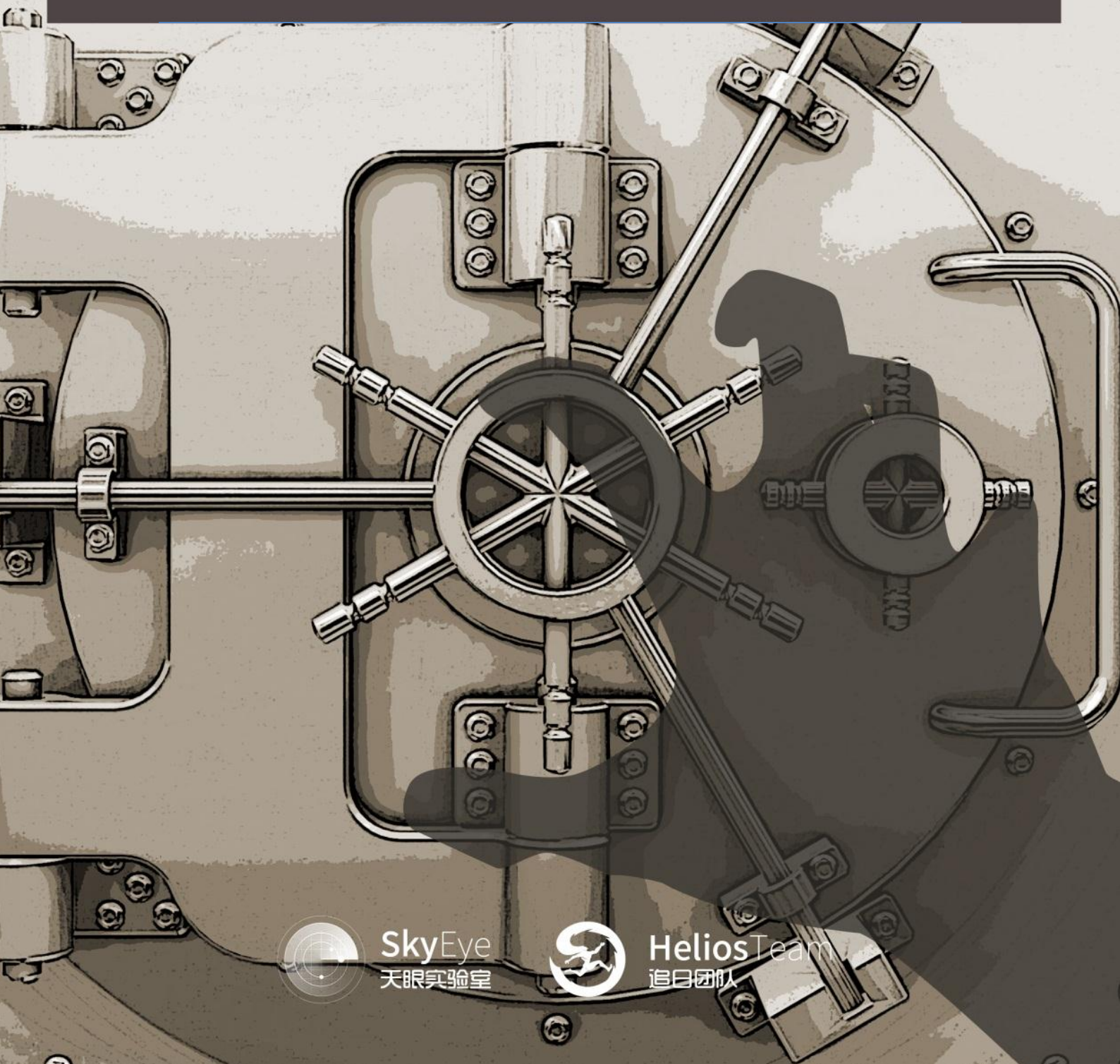
# APT

# 针对银行SWIFT系统攻击事件的综合分析报告



SkyEye 天眼实验室

HeliosTeam 追日团队

# 目录

| 报告更新相关时间节点 |
| --- |
| **2016 年 6 月 17 日，形成关联综合分析** |
| **2016 年 6 月 27 日，修改部分内容** |

# 一、 概述

2016 年 2 月孟加拉国央行被黑客攻击导致 8100 万美元被窃取的事件被曝光后，如越南先锋银行、厄瓜多尔银行等，针对银行 SWIFT 系统的其他网络攻击事件逐一被公开。在相关事件曝光后，我们立即对相关攻击事件的展示溯源分析，就越南先锋银行相关攻击样本，我们形成了技术报告：《SWIFT 之殇——针对越南先锋银行的黑客攻击技术初探》[1]。

在分析孟加拉国央行和越南先锋银行攻击事件期间，我们发现近期曝光的这 4 起针对银行的攻击事件并非孤立的，而很有可能是由一个组织或多个组织协同发动的不同攻击行动。另外通过对恶意代码同源性分析，我们可以确定本次针对孟加拉国央行和越南先锋银行的相关恶意代码与 Lazarus 组织（APT-C-26）有关联，但我们不确定幕后的攻击组织是 Lazarus 组织（APT-C-26）。

另外攻击组织对目标银行作业流程极为熟悉，也就并非短期内所能达到的，我们推测在侦查跟踪环节，攻击者应该针对目标进行了长时间且非常专注的持续性分析。

在对相关攻击事件的分析和剖析过程中，也暴露出诸多银行等金融行业本身的安全问题。如这一系列攻击事件要想达到金钱窃取，前提就需要获得银行本身 SWIFT 操作权限，而要获得相关权限则首先需要将银行自身网络攻陷。

近年来，针对银行、证券等金融行业的 APT 攻击不断出现，尽管目前披露的还只是以境外银行业发生的安全事件为主，但是网络攻击本就是跨国界的，这对于国内银行业的安全防护也敲响了警钟。在过去的安全实践中，我们不止一次发现了国内金融行业曾遭受到了 APT 攻击，安全态势并不是天下太平；再结合之前安天移动发布的《针对移动银行和金融支付的持续黑产行动披露——DarkMobileBank 跟踪分析报告》[2]中所披露的地下黑产针对金融行业最终用户的攻击现状，我们确实有必要 重新审视国内金融行业所面临的安全风险，以及在过去的安全规划与建设基础上创新思路，以应对不断出现的新兴威胁。

[1] http://bobao.360.cn/learning/detail/2890.html
[2] http://blog.avlsec.com/2016/04/3006/darkmobilebank/#list_2_3

# 二、 孟加拉央行攻击事件回顾

## 1. 背景

2016 年 2 月 5 日，孟加拉国央行（Bangladesh Central Bank）被黑客攻击导致 8100 万美元被窃取，攻击者通过网络攻击或者其他方式获得了孟加拉国央行 SWIFT 系统操作权限，进一步攻击者向纽约联邦储备银行（Federal Reserve Bank of New York）发送虚假的 SWIFT 转账指令，孟加拉国央行在纽约联邦储备银行上设有代理帐户。纽约联邦储备银行总共收到 35 笔，总价值 9.51 亿美元的转账要求，其中 30 笔被拒绝，另外 5 笔总价值 1.01 亿美元的交易被通过。进一步其中 2000 万美元因为拼写错误（Foundation 误写为 fandation）被中间行发觉而被找回，而另外 8100 万美元则被成功转走盗取。

而我们捕获到的这次网络攻击中所使用的恶意代码，其功能是篡改 SWIFT 报文和删除相关数据信息以掩饰其非法转账的痕迹，其中攻击者通过修改 SWIFT 的 Alliance Access 客户端软件的数据有效性验证指令，绕过相关验证。

## 2. 攻击流程

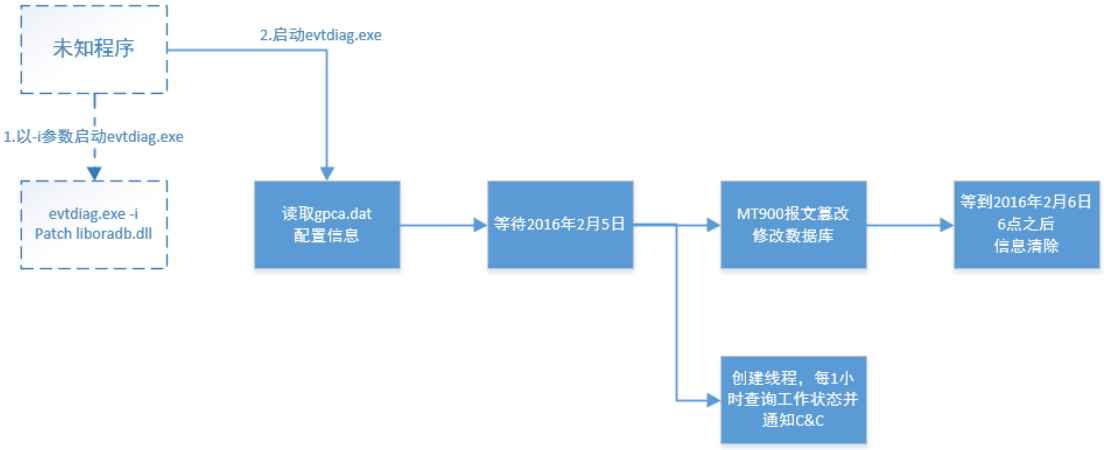



图 1 evtdiag.exe 执行流程

- 步骤 1：恶意代码检测是否有进程加载了“liboradb.dll”模块，进一步修改数据有效性验证指令，绕过验证；
- 步骤 2：读取“gpca.dat”配置文件，其中包括了 transord、日期、C&C 等攻击者预设的关键信息；
- 步骤 3：“2016 年 2 月 5 日”是样本在满足其他特定条件后，执行报文篡改操作的触发时间；
- 步骤 4：MT900 报文篡改，操作打印机，并选择性修改数据库；
- 步骤 5：样本执行篡改报文操作时，查询被感染计算机的相关“登录/注销”状态，将相关信息回传 C&C 服务器；
- 步骤 6：监控执行持续到 2016 年 2 月 6 日 6:00，之后退出并删除自身的日志、数据以及注册的服务。

| 选项 | 参数 | 说明 |
| --- | --- | --- |
| **无参数** | | 1. 配置信息读取(gpca.dat, mcm)<br>2. 等待 JRNL_20160205.JRNL_DISPLAY_TEXT 中出现 Login<br>3. swift 账单操作<br>4. 等待 20160206.06 清除操作信息,调用 evtsys.exe |
| **-svc** | 无 | 注册服务，执行程序主要流程 |
| **-p** | [PRINTER] resume | 恢复 PRINTER 的运行 |
| | [PRINTER] pause | 暂停 PRINTER 的运行 |
| | [PRINTER] on | 运行 PRINTER |
| | [PRINTER] off | 停止 PRINTER |
| | [PRINTER] queue | 枚举打印任务 |
| **-s** | [FILENAME] | 修改 SWIFT 目录下的 nroff.exe 为 rnoff.exe。将 [FILENAME] 重命名为 nroff.exe, 目的为使用 [FILENAME]劫持 nroff.exe |
| **-r** | 无 | 重命名 "nroff.exe" 为 "nroff.exe.bak", 重命名 "rnoff.exe"为"nroff.exe,目的是恢复 nroff 劫持 |
| **-t** | [CMD] | 连接 196.202.103.174:80, 通过 HTTP GET 向 "/al?"发送数据,参数为 [CMD] |
| **-i** | 无 | Patch Dll:遍历进程,更改 liboradb.dll 中 0x6A8B6 偏移处的内容为 9090 |
| **-u** | 无 | Unpatch Dll: 遍历进程, 更改 liboradb.dll 中 0x6A8B6 偏移处的内容为 7504 |
| **-g** | [DATE] | 查询 SWIFT 数据库 "JRNL_DATE" 表中的 "JRNL_DISPLAY_TEXT" 字段是否有 "Logout" 和 "Login"信息,输出显示 |

表 1 evtdiag.exe 相关参数

| 偏移量 | 数据 | 说明 |
| --- | --- | --- |
| **0x0** | 0A0B0C0Dh | Magic 标记 |
| **0x4** | 66h | transord 个数 |
| **0x8h - 0x8007** | 00901/0000058500<br>00901/0000058501<br>..... | 102 个 transord(gpca.dat 含有) |
| **0x8008** | 20160205 | 检查登录日期 |
| **0x8028** | D:\Alliance\Access\database\bin\sqlplus.exe | swift 客户端 sqlplus 路径 |
| **0x812C** | D:\MESSAGE_PARTNER | printer 相关路径 |
| **0x8230** | D:\Alliance\Access\common\bin\Win32 | -s,-r 操作的目录 |
| **0x8334** | 196.202.103.174 | 远程 C&C |

表 2 gpca.dat 配置文件内容

# 3. 篡改 MT900 报文

《SWIFT 之殇——针对越南先锋银行的黑客攻击技术初探》[3]中“二、 关于 SWIFT”，详细介绍了 SWIFT。MT900 是 SWIFT MT 十大类报文中其中第 9 类的一种，关于 MT900 报文的格式，下面有详细介绍，这样有助于理解后门具体篡改细节。

## 1) MT900 借记证实

### MT900 范围

这是由帐户行发给开户行，用来通知开户行某笔款项已借记其所开帐户的报文格式。该借记将在对帐单中被进一步证实。如果帐户行经常发送该帐户的对帐单，那么就不会定期发送此报文。

该报文不能用于记帐，它只是向收报行(即开户行)证实这一笔借记。

MT900 格式（M = Mandatory O = Optional）

| Status | Tag | Field Name | Content/Options |
|---|---|---|---|
| M | 20 | Transaction Reference Number | 16x |
| M | 21 | Related Reference | 16x |
| M | 25 | Account Identification | 35x |
| M | 32A | Value Date, Currency Code, Amount | 6!n3!a15d |
| O | 52a | Ordering Institution | A or D |
| O | 72 | Sender to Receiver Information | 6 * 35x |

### MT900 域详述[4]

- 域 20：发报行的编号
- 域 21：有关业务编号

  列明引起这笔借记的业务编号。如：MT100 中域“20”中的编号。
- 域 25：帐号

  列明已被借记的帐户号码。
- 域 32A：起息日、货币、金额

  列明借记的起息日、货币和金额。
- 域 52a：指示行

  列明指示发报行借记该帐户的银行。如果该银行是收报行以外的银行，那么报文使用该域列明指示行。
- 域 72：附言

  该域只能填写有关说明，不能出现任何指示。

[3] http://bobao.360.cn/learning/detail/2890.html

[4] https://ibanking.standardbank.com.na/corp/L001/helpfiles/Help_Files/Retail%20Balance%20and%20Transaction%20Information/Del/SWIFTMT940950.htm#MT950

## 2) 具体篡改流程

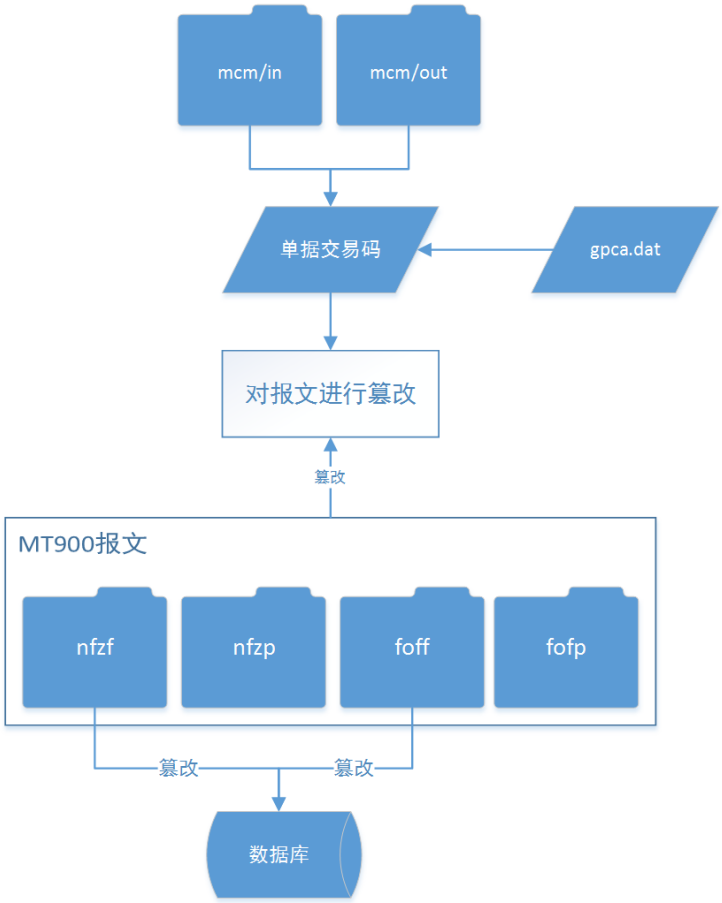



图 2 相关被篡改文件和配置文件关系图

获取 nfzp/nfzf 下所有"%d_%d"报文，并且根据 MSG_TRN_REF 是否已经在配置纪录当中进行分类，同时获取"%d_1"中的"60F"和"Sender"。

**注：以下表 3 和表 4 是 MT900 报文中具体需要修改的项，具体修改操作细节暂不公开**

| 项目 |
|---|
| **60F/60M:** |
| Debit/Credit |
| Amount |
| **61：** |
| **62F/62M:** |
| Debit/Credit |
| Amount |
| **64：** |
| Debit/Credit |
| Amount |

表 3 被篡改的 MT900 相关内容 1

此处操作的执行条件为：上张报文中的"Sender"为"FEDERAL RESERVE BANK"。foff 下会更新数据库，fofp 时不会跟新数据库

打开 foff/fofp 目录下的第一个含有"-"的报文。更该账单中的 19A 和 90B，在账单中，19A 和 90B 应该只有一项。

| 项目 |
| --- |
| **19A:**<br>**Amount** |
| Amount |
| **90B:**<br>**Price** |

表 4 被篡改的 MT900 相关内容 2

# 4. liboradb.dll 分析

本次事件中攻击者通过修改 SWIFT 的 Alliance Access 客户端软件中数据有效性验证指令，绕过相关验证，而导致相关问题的文件就是 liboradb.dll。

liboradb.dll 基于 OCI 编程开发，作为 SWIFT alliance 核心组件，为程序提供 Oracle 数据库交互操作接口，其中包含权限验证功能。此 dll 被 SWIFT 数据库服务进程调用,作为连接 Oracle 数据库的接口。

OCI 介绍：OCI（Oracle Call Interface，即 Oracle 调用层接口）是 Oracle 公司提供的由头文件和库函数等组成的一个访问 Oracle 数据库的应用程序编程接口（application programming interface API），它允许开发人员在第三代编程语言（包括 C, C++, COBOL 与 FORTRAN）中通过 SQL（Structure Query Language）来操纵 Oracle 数据库，而且 OCI 在一定程度上支持第三代编程语言（诸如 C, C++, COBOL 与 FORTRAN）的数据类型、语法等等。

# 三、 相关攻击事件综合分析

## 1. SWIFT 官方预警或申明

2016 年 5 月 9 日，环球银行金融电信协会（SWIFT）发表声明表示[5]，SWIFT 拒绝由孟加拉国银行和孟加拉国警方的刑事调查部门(CID)官员提出的虚假指控，SWIFT 对孟加拉银行劫案不负有任何责任，孟加拉银行有责任维护其银行系统环境的安全。

2016 年 5 月 10 日，孟加拉国央行的新掌门人、纽约联邦储备银行和 SWIFT 官员在瑞士巴塞尔会晤讨论。在一份简短的联合声明中，双方表示，他们致力于追回被窃资金，将肇事者绳之以法，并协同工作来“标准化操作”。[6]

2016 年 5 月 13 日，SWIFT 协会发布的一份报告[7]中称，已有第二家银行报告遭到网络攻击，这起攻击与孟加拉国央行在纽约联邦储备银行的账户被窃 8100 万美元的网络攻击类似，具体损失情况未知。并强调称，针对孟加拉国央行的恶意软件，对 SWIFT 的网络或核心信息交互系统没有影响，该恶意软件只能在黑客已经成功发现并利用当地（银行）系统网络安全隐患之后才能被植入。就此，SWIFT 已经研发出相应设备，帮助客户提升网络安全、找准当地数据库记录有出入之处。

2016 年 5 月 24 日，在布鲁塞尔欧洲金融服务第十四届年度会议上，SWIFT 首席执行官 Gottfried Leibbrandt 表示[8]，SWIFT 将提升其网络系统安全性，采取包括对银行管理软件提出更严格的安全要求，管控支付方式和第三方机构认证等措施。他重申，攻击并未对 SWIFT 的网络或核心信息交互系统造成影响，并透露将会启动一个新项目，旨在维护全球金融体系安全。

2016 年 5 月 27 日，SWIFT 协会宣称启动新的客户项目，针对日益猖獗的网络威胁，保护全球金融体系的财产安全。该项目分为 5 个战略举措[9]，包括提高国际机构之间信息共享、增强客户的 SWIFT 相关工具、加强指导，提供审计的框架、支持增加事务模式检测、加强支持第三方提供者。

## 2. 战术：瞄准 SWIFT 系统

### 1) 针对银行系统的一系列攻击事件

### 2016 年-孟加拉国央行（Bangladesh Central Bank）

在本报告第二部分内容详细介绍了攻击流程和篡改 MT900 报文的细节，在这里不进一步占据，具体内容请参看：“二、孟加拉央行攻击事件回顾”。

---

[5] https://www.swift.com/insights/press-releases/swift-statement
[6] https://www.newyorkfed.org/newsevents/statements/2016/0510-2016
[7] https://www.swift.com/insights/press-releases/swift-customer-communication_customer-security-issues
[8] https://www.swift.com/insights/press-releases/gottfried-leibbrandt-on-cyber-security-and-innovation
[9] https://www.swift.com/insights/press-releases/swift-launches-customer-security-programme-to-reinforce-the-security-of-the-global-banking-system

## 2015 年-越南先锋银行（Tien Phong Bank）


图 3 整体关系流程

针对越南先锋银行的攻击中，相关恶意代码内置了 8 家银行的 SWIFT CODE，越南银行均在这些银行中设有代理帐户。目前看到的 Fake PDF Reader 样本目的不是攻击列表中的这些银行，而是用来删除越南银行与其他家银行间的转帐确认（篡改 MT950 对帐单）。这样银行的监测系统就不会发现这种不当交易了。

关于针对越南先锋银行的攻击，可以参看我们之前发布的报告：《SWIFT 之殇——针对越南先锋银行的黑客攻击技术初探》[10]。

## 2015 年-厄瓜多尔银行（Banco del Austro）[11]

据路透社报道，2015 年 1 月 12 号，在一条来自厄瓜多尔 Banco del Austro(DBA)银行系统信息的指引下，位于旧金山的 Wells Forga 向香港的银行账户进行了转账。并且在接连 10 天内，至少有 12 笔的 BDA 银行资金通过 SWIFT 系统被转走，总金额高达 1200 万美金。BDA

[10] http://bobao.360.cn/learning/detail/2890.html
[11] http://www.reuters.com/article/cyber-heist-swift-idUSL2N18H04S

已就该事件将 Wells Frago 向纽约法庭提起了诉讼，理由是 Wells Forgo 美国银行本应该将这些交易标记为可疑交易，然而从诉讼资料看，双方银行都相信这些资金是被匿名黑客盗走的。

另外，SWIFT 方面的负责人在案件被报道之前却对此毫不知情。相关人士称，SWIFT 确实会核验系统发送信息中的密码来确保信息来自银行用户的终端设备。但是一旦网络盗窃者获取了密码和证书，SWIFT 就无法判断操作者是不是真正的账户持有人了。而黑客正式钻了这个空子，盗取了一名银行雇员的 SWIFT 证书，金额盗走了巨额资金。

## 2013 年-索纳莉银行（Sonali Bank）[12]

据路透社报道，2013 年孟加拉国的索纳莉银行（Sonali Bank）也发生了类似孟加拉央行的攻击事件，在索纳莉事件中，攻击者盗取了 25 万美金的银行资金。银行 IT 运营部的高级官员称，在索纳莉银行劫案中，黑客们在一台电脑上安装 keylogger 来窃取其他系统的密码，然后使用 SWIFT 系统发送伪造的转账申请。

# 2) 相似的攻击战术

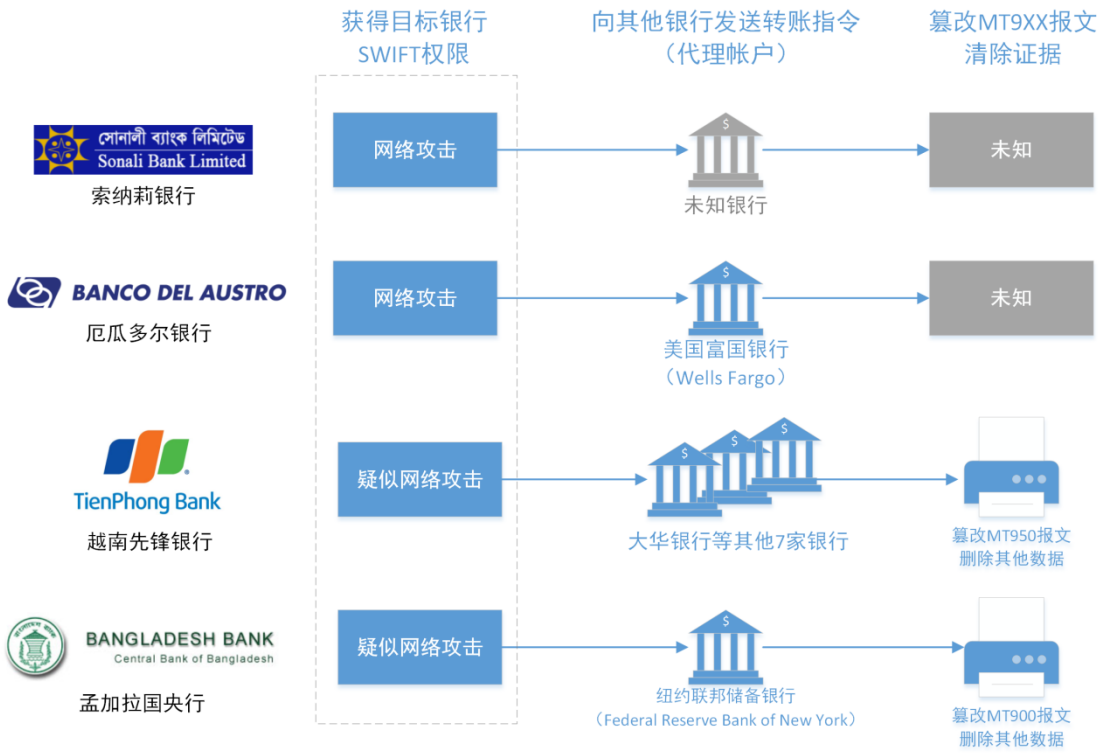



图 4 关于针对 4 家银行的攻击流程

通过分析从 2013 年的索纳莉到 2016 年的孟加拉国央行这 4 个攻击银行的事件，结合上图，不难看出相关攻击事件之间有很多的相似性。

从攻击战术或攻击流程进行分析，其中主要环节是获得 SWIFT、利用 SWIFT 发送转账指令和最终清除证据掩盖事实这三个部分。

[12] http://in.reuters.com/article/cyber-heist-bangladesh-exclusive-idINKCN0YG2VV

**第一，获得目标银行 SWIFT 权限：**首先需要获得目标银行的 SWIFT 系统操作权限，索纳莉银行和厄瓜多尔银行从相关报道来看，均是通过网络攻击来获得了相关权限。据有关报道称索纳莉银行 SWIFT 相关登录帐号和密码是被植入的恶意代码所监控窃取，而厄瓜多尔也是类似登录验证凭证被窃取，由此我们可以得到一个信息，就是攻击者要获得 SWIFT 操作权限，并不需要进行物理接触，完全通过网络即可完成。而目前尚未有报道明确指出孟加拉国央行的 SWIFT 系统权限是通过网络攻击获得，但相关调查孟加拉央行事件的研究人员表示应该是黑客利用网络攻击获得了相关登录凭证。而越南先锋银行本身没有被攻击，问题出在其第三方服务商（提供 SWIFT 服务），但目前不清楚是否是通过网络进行攻击获得相关 SWIFT 操作权限的，先锋银行之后表示要改为直接连接 SWIFT 系统。

**第二，向其他银行发送转账指令（代理帐户）：**攻击者在获得 SWIFT 权限之后，最核心的目的就是要利用 SWIFT 发送转账指令，我们推测应该是 SWIFT MT 报文中第一类报文，如 MT103（单笔客户汇款）。除索纳莉银行以外，我们发现攻击者均向存在目标银行代理帐户的银行发送转账指令，如美国国富银行设有厄瓜多尔银行的代理帐户、大华银行等其他 7 家银行设有越南先锋银行的代理帐户和纽约联邦储备银行设有孟加拉国央行的代理帐户。通俗来讲也就是孟加拉国央行等这几个目标银行存在其他银行上的钱被冒名转走了。

**第三，篡改 MT9XX 报文\清除证据：**由于暂未捕获到针对索纳莉和厄瓜多尔银行进行攻击的恶意代码，所以我们无法知道是否有该环节。我们主要来看越南先锋银行和孟加拉国央行，首先攻击者都是对 MT9XX 报文进行了劫持，在对越南先锋银行是劫持 MT950 对帐单，在针对孟加拉国央行是劫持了 MT900 借记证实，进一步都是对相关报文进行了篡改，目的是删除相关转帐记录，进行平帐。有区别是孟加拉国央行是对相关报文篡改后直接发送给打印机，打印出来，而越南先锋银行是对 MT950 的电子版 PDF 进行篡改后，然后打印机打印篡改后的 PDF。攻击者最终目的就是篡改报告，另外删除其他一些数据信息，目的是抹去相关证据线索。另外我们发现越南先锋银行和孟加拉国央行中攻击者所使用的恶意代码，从代码同源性上，都存在一个特殊的安全删除函数，这也更进一步证明的这两次攻击事件并不是孤立的，两者之间必然有一定联系。

总体而言，这类攻击战术就是针对具备 SWIFT 系统的银行进行攻击，首先需要依托网络攻击或其他手段获得 SWIFT 权限，进一步向其他银行上的代理帐户发送转帐指令来达到金钱窃取的目的，最终使用恶意代码进行相关证据清除掩盖事实的过程。

# 3. 技术：恶意代码同源性分析

## 安全删除函数

我们发现孟加拉国央行、越南先锋银行攻击中使用的恶意代码内的安全删除函数是复用了相同的代码，进一步 Lazarus 组织在 2014 年针对索尼的攻击中的恶意代码和赛门铁克安全公司在近期发布的安全报告[13]中提到，针对东南亚金融业的有限目标性攻击中出现的 Backdoor.Fimlis 恶意代码，都复用了同样的代码。

安全删除函数有 2 个参数：文件名和迭代次数。首先使用 5F 覆盖文件的末尾字节，然后根据 6 个控制字节决定使用什么数据覆盖原文件内容。

| 孟加拉国央行 | 越南先锋银行 |
| --- | --- |

[13] http://www.symantec.com/connect/blogs/swift-attackers-malware-linked-more-financial-attacks

```
v1 = CreateFileA(lpFileName, 0x40000000u, 0, 0, 3u,
v2 = v1;
if ( v1 == (HANDLE)-1 )
{
  result = GetLastError();
}
else
{
  SetFilePointer(v1, -1, 0, 2u);
  WriteFile(v2, &Buffer, 1u, &NumberOfBytesWritten,
  FlushFileBuffers(v2);
  FileSize = 0i64;
  GetFileSizeEx(v2, &FileSize);
  SetFilePointer(v2, 0, 0, 0);
  v4 = FileSize.HighPart;
  v5 = FileSize.LowPart;
  v6 = 0;
  v7 = 0;
  if ( FileSize.HighPart >= 0 && (FileSize.HighPart
  {
    while ( 1 )
    {
      v8 = __OFSUB__(__PAIR__(v4, v5), __PAIR__(v7,
      v11 = v5 - v6;
      v9 = (__PAIR__(v4, v5) - __PAIR__((unsigned i
      v10 = v5 - v6;
      if ( v9 < 0 || (unsigned __int8)((v9 < 0) ^ v
      {
        v15 = v9;
      }
      else
      {
        v10 = 4096;
        v15 = 0;
      }
      if ( !WriteFile(v2, &Buffer, v10, &NumberOfBy
        break;
```

```
v3 = CreateFileA(lpFileName, 0x40000000u, 0
v4 = v3;
if ( v3 == (HANDLE)-1 )
{
  result = GetLastError();
}
else
{
  SetFilePointer(v3, -1, 0, 2u);
  WriteFile(v4, &Buffer, 1u, &NumberOfBytes
  FlushFileBuffers(v4);
  GetFileSizeEx(v4, &FileSize);
  v6 = 0;
  for ( i = 0; ; v6 = i )
  {
    v7 = a2;
    if ( a2 > 6 )
      v7 = 6;
    if ( v6 >= v7 )
      break;
    SetFilePointer(v4, 0, 0, 0);
    if ( *(&v17 + v6) == -1 )
    {
      sub_40D3A0(&Buffer, 4096);
    }
    else
    {
      LOBYTE(v8) = *(&v17 + v6);
      BYTE1(v8) = *(&v17 + v6);
      v9 = v8 << 16;
      LOWORD(v9) = v8;
      memset32(&Buffer, v9, 0x400u);
    }
    v10 = FileSize.HighPart;
    v11 = FileSize.LowPart;
    v12 = 0i64;
```

```
memset(&v8, 0, 0x100u);
v9 = 0;
v10 = 0;
strcpy(&Str, lpExistingFileName);
v3 = strrchr(&Str, 92);
if ( v3 )
  v4 = v3 + 1;
else
  v4 = &Str;
if ( *v4 )
{
  do
  {
    *v4 = rand() % 26 + 97;
    v5 = (v4++)[1];
  }
  while ( v5 );
}
if ( MoveFileA(lpExistingFileName, &Str)
  v2 = &Str;
if ( a2 )
{
  if ( !RemoveDirectoryA((LPCSTR)v2) )
    return GetLastError();
}
else if ( !DeleteFileA((LPCSTR)v2) )
{
  return GetLastError();
}
```

```
memset(&v8, 0, 0x100u);
v9 = 0;
v10 = 0;
strcpy(&Str, lpExistingFileName);
v3 = strrchr(&Str, '\\');
if ( v3 )
  v4 = v3 + 1;
else
  v4 = &Str;
if ( *v4 )
{
  do
  {
    *v4 = rand() % 26 + 97;
    v5 = (v4++)[1];
  }
  while ( v5 );
}
if ( MoveFileA(lpExistingFileName, &Str) )
  v2 = &Str;
if ( a2 )
{
  if ( !RemoveDirectoryA((LPCSTR)v2) )
    return GetLastError();
}
else if ( !DeleteFileA((LPCSTR)v2) )
{
  return GetLastError();
}
return 0;
```

| Lazarus 组织 | Backdoor.Fimlis |
| --- | --- |

```
v3 = CreateFileA(Source, 0x40000000, 0, 0, 3,
v4 = v3;
v21 = v3;
if ( v3 == -1 )
{
  result = GetLastError();
}
else
{
  SetFilePointer(v3, -1, 0, 2);
  WriteFile(v4, &Dst, 1, &v22, 0);
  FlushFileBuffers(v4);
  GetFileSizeEx(v4, v19);
  v20 = 0;
  while ( 1 )
  {
    v6 = a2;
    if ( a2 > 6 )
      v6 = 6;
    v7 = v20;
    if ( v20 >= v6 )
      break;
    SetFilePointer(v4, 0, 0, 0);
    v8 = *(&v23 + v7);
    if ( v8 == -1 )
      sub_401C82(&Dst, 4096);
    else
      memset(&Dst, v8, 0x1000u);
    v9 = v19[1];
    for ( i = 0i64; i < *(_QWORD *)v19; i +=
```

```
v8 = CreateFileW(a3, 0x40000000, 0, 0, 3,
v9 = v8;
if ( v8 == -1 )
{
  result = GetLastError();
}
else
{
  SetFilePointer(v8, -1, 0, 2, a2, a1);
  WriteFile(v9, &v28, 1, v24, 0);
  FlushFileBuffers(v9);
  SetFilePointer(v9, 0, 0, 0, v20, v21);
  GetFileSizeEx(v9, &v23);
  for ( i = 0; ; ++i )
  {
    v13 = a6;
    if ( a6 > 6 )
      v13 = 6;
    if ( i >= v13 )
      break;
    if ( v22[i] == -1 )
    {
      sub_401540(&v28, 4096);
    }
    else
    {
      LOBYTE(v11) = v22[i];
      BYTE1(v11) = v22[i];
      v14 = v11 << 16;
      LOWORD(v14) = v11;
      memset32(&v28, v14, 0x400u);
    }
    v15 = 0;
    v16 = 0;
```

```
  memset(&v7, 0, 0x100u);
  v8 = 0;
  v9 = 0;
  v2 = Source;
  strcpy(&Dest, Source);
  v3 = strrchr(&Dest, 92);
  if ( !v3 )
  {
    v3 = &Dest;
    goto LABEL_4;
  }
  while ( 1 )
  {
    ++v3;
LABEL_4:
    if ( !*v3 )
      break;
    *v3 = rand() % 26 + 97;
  }
  if ( MoveFileA(Source, &Dest) )
    v2 = &Dest;
  if ( a2 )
    v4 = RemoveDirectoryA(v2);
  else
    v4 = DeleteFileA(v2);
  if ( v4 )
    result = 0;
  else
    result = GetLastError();
  return result;
}
```

```
  CloseHandle(v9);
  wcscpy(&Dest, Source);
  v18 = wcsrchr(&Dest, '\\');
  if ( v18 )
    v19 = v18 + 1;
  else
    v19 = &Dest;
  for ( ; *v19; *(v19 - 1) = rand() % 26 + 97 )
    ++v19;
  if ( MoveFileW(Source) )
    wcscpy(Source, v24);
  if ( dword_404798() )
    result = 0;
  else
    result = GetLastError();
}
return result;
```

# 四、 总结

## 1. 攻击由一个组织或多个组织协同发起

从对相关攻击事件的战术层面和技术层面的深入分析，我们认为近期曝光的这 4 起针对银行的攻击事件并非孤立的，而很有可能是由一个组织或多个组织协同发动的不同攻击行动。

## 2. 攻击组织极为熟悉目标银行的作业流程

如越南先锋银行中，从将恶意程序构造伪装成 Foxit reader（福昕 PDF 阅读器）到对 MT950 对帐单 PDF 文件的解析和精确的篡改等攻击手法，都反映出攻击者对银行内部交易系统和作业流程非常熟悉。攻击者的攻击意图明确，而且攻击者要如此了解和展开相关攻击行动，事前进行了大量侦查情报收集的工作。

## 3. 与 Lazarus 组织（APT-C-26）存在关联

针对 SWIFT 攻击事件中与 Lazarus 组织所使用的相关恶意代码，我们从样本代码层面进行同源性分析，发现其中一个特殊的安全删除函数基本是进行了代码复用。从这一点来看，针对越南先锋银行和孟加拉国央行的攻击应该是与 Lazarus 组织有一定的联系。

安全删除函数这部分代码能关联到 Lazarus 组织曾今在 2013 年发动的 darkseoul 攻击行动和 2014 年针对索尼影视娱乐公司的攻击行动，相关攻击行动的 IOC（MD5\C&C 等）在当时已经被安全机构公开了，也可以理解为安全删除函数这个本身特殊的代码在当时就已经公开了。也就是在此之后，比如 2015 年、2016 年非 Lazarus 组织的攻击者，也可以轻松的获得安全删除函数的代码并在进行开发其他恶意代码的时候拿来使用。简而言之，如果我们依靠这处安全删除函数，来判定某个恶意代码是否属于 Lazarus 组织，是不具备强关联性的。

正如我们之前发布的洋葱狗报告（APT-C-03）[14]“第 5 章 ICEFOG‘重生’：误导？嫁祸？”中提到的观点，我们不排除这有可能是其他组织刻意加入的干扰项。

## 4. 银行业本身暴露出诸多安全问题

近期曝光的 4 起针对银行的攻击事件中，其中 2013 年的索纳莉银行、2015 厄瓜多尔银行确定是由网络进行攻击获得相关转账权限，另外越南先锋银行和孟加拉国央行也是自身环节发生了问题，导致攻击者具备发送 SWIFT 转账指令的权限。

这明显暴露出银行自身的安全防护薄弱，另外攻击者通过网络攻击就可以获得 SWIFT 权限，并加以操作，以及攻击者对 SWIFT 的 Alliance Access 客户端软件的数据有效性验证指令，绕过相关验证等等，这些都暴露出 SWIFT 本身也存在一定问题，如是否在普通的帐号密码验证机制基础上，可以加一些需要依赖物理设备或环境才能进行验证的步骤，这样能大大隔离纯粹来自网络的攻击。

[14] https://ti.360.com/upload/report/file/Operation_OnionDog.pdf