Extracting actionable data from modern financially motivated malware on Windows personal computers

James Wyke

Technical Report

RHUL–ISG–2016–14

5 April 2016
Extracting Actionable Data from Modern Financially Motivated Malware on Windows Personal Computers

By

James Wyke
Executive Summary

In this report we describe the wide range of techniques used by financial malware to facilitate online banking fraud. It describes how financial malware has evolved over the last decade to become more efficient and effective at facilitating fraud.

We describe actionable data that can be extracted from financial malware and used to remediate existing and protect against future compromises. We assess how that data can be extracted and provide a working solution for the Vawtrak malware family that can be used and extended by Incident Responders, forensic and malware analysts.
# Table of Contents

- Executive Summary ................................................................. 3
- Table of Contents ........................................................................ 4
- Figures and Tables ........................................................................ 8
- Abbreviations and Acronyms ....................................................... 9

1. Introduction .................................................................................. 10
   1.1 Objectives .............................................................................. 10
   1.2 Scope .................................................................................... 11
   1.3 Definitions and Background .................................................... 12
   1.4 Malware Families ................................................................... 13

2. Financially Motivated Malware ..................................................... 14
   2.1 Data Capture ......................................................................... 14
      2.1.1 Keystroke Loggers .............................................................. 15
         2.1.1.1 SetWindowsHook ....................................................... 15
         2.1.1.2 Window-based APIs .................................................. 16
      2.1.3 Interface Replacement ..................................................... 16
         2.1.3.1 Website .................................................................... 16
         2.1.3.2 Banking Application ................................................ 17
      2.1.4 Man-in-the-Browser ....................................................... 17
         2.1.4.1 Browser Extensions ................................................... 18
         2.1.4.2 API Hooking ............................................................ 19
      2.1.5 Screen Capture .................................................................. 22
         2.1.5.1 Screenshots ............................................................... 22
         2.1.5.2 Video ...................................................................... 23
         2.1.5.3 Event-driven ............................................................. 23
      2.1.6 Clipboard Capture ............................................................ 23
      2.1.7 Stored Credential Theft ................................................... 24
         2.1.7.1 Browsers ................................................................. 24
         2.1.7.2 Email Clients ........................................................... 25
         2.1.7.3 FTP Clients ............................................................. 25
         2.1.7.4 Cookies ................................................................. 26
      2.1.8 Form Grabbing ............................................................... 26
      2.1.9 Local Proxy ..................................................................... 27
2.1.10  Techniques Comparison .......................................................................................................................... 28
2.2  Data Exfiltration ................................................................................................................................................. 29
  2.2.1  HTTP ............................................................................................................................................................... 30
     2.2.1.1  Plain Text ................................................................................................................................................ 30
     2.2.1.2  Weak Encryption .................................................................................................................................. 30
     2.2.1.3  Symmetric Key Encryption .................................................................................................................. 31
     2.2.1.4  Public Key Encryption ......................................................................................................................... 32
  2.2.2  HTTPS ............................................................................................................................................................. 32
  2.2.3  FTP ................................................................................................................................................................. 33
  2.2.4  SMTP ............................................................................................................................................................... 33
  2.2.5  Custom Protocols ......................................................................................................................................... 34
  2.2.6  Peer-to-Peer .................................................................................................................................................. 34
  2.2.7  Tunneling ....................................................................................................................................................... 35
  2.2.8  The Darknet .................................................................................................................................................. 35
     2.2.8.1  Tor ......................................................................................................................................................... 35
     2.2.8.2  I2P ........................................................................................................................................................... 36
  2.2.9  Domain Generation Algorithms ................................................................................................................ 37
  2.2.10 Multiple Fall-back Options ....................................................................................................................... 37
2.3  Stealth ................................................................................................................................................................. 37
  2.3.2  User-mode Rootkits .................................................................................................................................. 38
  2.3.3  Kernel-mode Rootkits ................................................................................................................................ 38
2.4  Data Modification ............................................................................................................................................. 38
  2.4.1  Web Page Code Injection ........................................................................................................................... 39
     2.4.1.1  Extra Fields .............................................................................................................................................. 39
     2.4.1.2  Remove Warnings ................................................................................................................................ 40
     2.4.1.3  Dynamic Injection ................................................................................................................................ 40
  2.4.4  Clipboard Modification ................................................................................................................................ 41
  2.4.5  DNS Redirection ......................................................................................................................................... 41
     2.4.5.1  Hosts File .................................................................................................................................................. 42
     2.4.5.2  API Hooks ............................................................................................................................................... 42
2.5  Automation ......................................................................................................................................................... 42
  2.5.1  Automated Alerts ......................................................................................................................................... 43
  2.5.2  Automatic Transfer System ....................................................................................................................... 43
2.6  Defeating Countermeasures ............................................................................................................................ 43
2.6.1 Countermeasures ........................................................................................................ 44
  2.6.1.1 Antivirus .................................................................................................................. 44
  2.6.1.2 Browser Protection ................................................................................................. 44
  2.6.1.3 Two-Factor Authentication ..................................................................................... 44
  2.6.1.4 Fraud Detection ....................................................................................................... 45
2.6.2 Workarounds ............................................................................................................. 45
  2.6.2.1 One-time Password Interception ............................................................................ 45
  2.6.2.2 Mobile Components ............................................................................................. 46
  2.6.2.3 Anti-Anti-Virus ...................................................................................................... 46
  2.6.2.4 Remote Desktop .................................................................................................... 46
2.7 Modularisation ............................................................................................................. 47
3 Extracting Actionable Data ............................................................................................. 48
  3.1 Defining Actionable Data ............................................................................................ 48
    3.1.1 Directly Actionable ................................................................................................. 49
    3.1.2 Intermediary Data .................................................................................................. 49
  3.2 Relevance of Actionable Data ..................................................................................... 50
    3.2.1 Aiding Investigation ............................................................................................... 50
    3.2.2 Further Compromise .............................................................................................. 50
  3.3 Memory ....................................................................................................................... 51
    3.3.1 String Searching ..................................................................................................... 51
    3.3.2 Artefact Carving ..................................................................................................... 52
    3.3.3 Memory Analysis Suites ....................................................................................... 52
      3.3.3.1 Volatility ............................................................................................................ 52
      3.3.3.2 Redline ............................................................................................................. 53
      3.3.3.3 HBGary Responder .......................................................................................... 53
  3.4 Network ....................................................................................................................... 53
    3.4.1 Manual Analysis ..................................................................................................... 54
    3.4.3 Automated Protocol Analysis ................................................................................. 54
      3.4.3.1 Custom Tools ..................................................................................................... 54
      3.4.3.2 Chopshop ......................................................................................................... 55
  3.5 Disk ............................................................................................................................... 55
    3.5.1 File System ............................................................................................................ 55
    3.5.2 Windows Registry .................................................................................................. 56
    3.5.3 Unallocated Blocks ................................................................................................. 57
Demonstrating Data Extraction ................................................................. 58

4.1 Malware Selection .................................................................................. 58

4.1.1 Vawtrak Overview.............................................................................. 58

4.1.2 Extraction Targets ............................................................................. 59

4.2 Reverse Engineering .............................................................................. 59

4.2.1 Dynamic Analysis................................................................................ 60

4.2.2 Unpacking ......................................................................................... 62

4.2.3 Finding Relevant Code ...................................................................... 63

4.3 Volatility Module ................................................................................... 66

4.3.1 Previous Work ................................................................................... 66

4.3.2 Vawtrakscan .................................................................................... 66

4.4 Chopshop Module .................................................................................. 68

4.4.1 Previous Work ................................................................................... 68

4.4.2 Vawtrak Module ............................................................................... 68

4.5 The Sleuth Kit ....................................................................................... 70

4.5.1 Extracting Files with TSK ................................................................ 70

4.6 Putting it all Together ........................................................................... 70

4.6.1 Design ............................................................................................... 70

4.6.2 Using ................................................................................................ 72

4.6.3 Output ............................................................................................... 72

4.6.4 Effectiveness ..................................................................................... 74

5 Conclusion .............................................................................................. 75

5.1 Future Work ......................................................................................... 75

Bibliography ............................................................................................... 77

Appendices ................................................................................................. 83

Appendix A ................................................................................................. 83

Appendix B ................................................................................................. 83
Figures and Tables

The following figures and tables have been used in this report. Where the figures have been taken from an external source, the reference is included. In all other cases the figures have been produced by the author.

Table 1 – data capture techniques comparison

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>figure 1</td>
<td>process injection sequence</td>
<td>[BA13]</td>
</tr>
<tr>
<td>figure 2</td>
<td>loaded dlls</td>
<td></td>
</tr>
<tr>
<td>figure 3</td>
<td>tor hidden services</td>
<td>[TP00]</td>
</tr>
<tr>
<td>figure 4</td>
<td>social security number added field</td>
<td>[RP10]</td>
</tr>
<tr>
<td>figure 5</td>
<td>vawtrak post request</td>
<td></td>
</tr>
<tr>
<td>figure 6</td>
<td>packed sample strings</td>
<td></td>
</tr>
<tr>
<td>figure 7</td>
<td>unpacked sample strings</td>
<td></td>
</tr>
<tr>
<td>figure 8</td>
<td>disassembled view</td>
<td></td>
</tr>
<tr>
<td>figure 9</td>
<td>network strings</td>
<td></td>
</tr>
<tr>
<td>figure 10</td>
<td>post reference</td>
<td></td>
</tr>
<tr>
<td>figure 11</td>
<td>vawtraksan snippet</td>
<td></td>
</tr>
<tr>
<td>figure 12</td>
<td>vawtraksan output</td>
<td></td>
</tr>
<tr>
<td>figure 13</td>
<td>cmdcodes processing</td>
<td></td>
</tr>
<tr>
<td>figure 14</td>
<td>sample config</td>
<td></td>
</tr>
<tr>
<td>figure 15</td>
<td>extract-py usage message</td>
<td></td>
</tr>
<tr>
<td>figure 16</td>
<td>extracted data from memory</td>
<td></td>
</tr>
<tr>
<td>figure 17</td>
<td>extracted cmds received</td>
<td></td>
</tr>
<tr>
<td>figure 18</td>
<td>decoded config</td>
<td></td>
</tr>
</tbody>
</table>
Abbreviations and Acronyms

This section provides some acronyms and abbreviations used in this report:

- MitB - Man-in-the-Browser
- BHO - Browser Helper Object
- TSK - The Sleuth Kit
- ATS - Automatic Transfer System
- API - Application Programming Interface
- FTP - File Transfer Protocol
- SMTP - Simple Mail Transfer Protocol
- HTTP - HyperText Transfer Protocol
- HTTPS - HTTP Secure
- TOR - The Onion Router
- I2P - Invisible Internet Protocol
- DGA - Domain Generation Algorithm
- DNS - Domain Name System
- JS - JavaScript
- PoS - Point of Sale
1. Introduction

Online banking fraud is a growing problem both in the UK and globally. According to [KP15], losses from online banking fraud in the UK alone rose by 48% in 2014 to over sixty million pounds. A large proportion of these losses are facilitated by financial malware.

This kind of malware has been in existence for over a decade and during that time its sophistication and effectiveness has vastly increased. The techniques used by modern malware and those behind the malware are very different to those used ten years ago.

Combating this kind of malware is not a simple task. Gaining as much knowledge as possible about a threat is an important component of that task. To that end this project attempts to better understand financial malware and identify what information we need to know to combat that malware and how we can obtain it.

In part one of this project we attempt to understand financial malware by making a detailed study of the techniques used to achieve the various functions such malware must carry out to enable online banking fraud.

In part two we explore the types of information that will help us combat financial malware and how we can extract that information.

In part three we give a practical demonstration that uses the knowledge we have gained to develop a toolset to help us combat financial malware.

1.1 Objectives

The objectives of this project are as follows:

1. Produce a history of the techniques used by financially motivated malware on Windows personal computers. This will focus on the techniques used by this category of malware to achieve its goals, ranging from early families through to modern day variants.

   We will examine the techniques used to capture data, which range from simple keystroke logging to man-in-the-browser techniques, video recording and cached credential theft; to exfiltrate data which can range from simple plain text over HTTP to custom peer-to-peer protocols; to stealth the presence of the malware on an infected computer; to modify data so that more valuable information can be stolen; to automate various aspects of the information theft and defrauding process; and to defeat the various countermeasures introduced by banks and security companies.
Through understanding how financial malware works we gain a far more informed insight into how we might develop solutions to extract the kind of data we are interested in.

2. Define and give an overview of the different types of actionable data that can be extracted, the methods used to do so, and their relevance, both to potential criminal investigations and to helping prevent future compromises.

We will establish exactly what makes data actionable and how we can go about extracting it from the data sources available to us. These data sources will be memory, the network and the disk.

This gives us a clear idea of what kinds of data we should aim to extract and to what purpose extracted data can be put.

3. Produce a practical demonstration of data extraction for one (or more if time allows) malware family.

We will provide a practical demonstration of data extraction by implementing stand-alone modules to extract data from memory and the network for a particular malware family. Furthermore we will implement a framework that will link the independent modules to build up a complete picture of the activity of the malware and enhance the data that we extract.

The stand-alone modules will provide value in their own right, as they can be directly used by incident responders, forensic analysts and network defenders to protect against the threat and to dissect the implications of a successful compromise. The framework that will bring the modules together streamlines the process and enriches the data by using intermediate data extracted from one source to extract further data from another source. As an example, we might extract a decryption key from memory and use it to decrypt records of data exfiltrated over the network.

1.2 Scope

This project covers financial malware on Windows desktop PCs. It does not cover malware on mobile devices, on non-Windows operating systems or on devices such as ATMs. It does not cover categories of malware other than financial malware.

Part one covers the techniques used by financial malware. It is not intended to be a comprehensive history of all financial malware that has ever existed.
Part two discusses actionable data and how to extract it. It is not a guide to incident response, forensic or malware analysis. Neither is it a comprehensive survey of forensic analysis tools and techniques.

Part three demonstrates data extraction for a particular malware family. During that demonstration we touch on reverse engineering the malware family in question, however, this is not a comprehensive analysis of the reverse engineering process.

1.3 Definitions and Background

This section provides definitions for the major terminology used in this report.

Modern financially motivated malware on personal computers running Windows

This is a subclass of malware that is found on home and corporate desktop PC’s. It is motivated by financial gain, and attempts to achieve that goal by stealing credentials to online banking applications or by obtaining unauthorised control of the victim’s bank account through other means such as session hijacking, automated transfers or two factor authentication bypasses.

Actionable data

This is data associated with a threat that can be directly used to further understand the nature of the threat itself, what actions it may have carried out, the identity of those who deployed it and how we might track the threat and find separate instances of it. This commonly includes network addresses used for control channels, decryption keys and downloaded configuration data.

Actionable data can be extremely useful for a variety of reasons:

- It can be used to aid criminal investigations by helping track malware campaigns and identify additional victims.
- It can be used to understand exactly what activity was carried out or what data may have been stolen, by using it to decrypt network communications between an infected machine and its control server. Data such as command and control addresses can be used by network defenders as indicators of compromise to find infections on their networks.

Attacker

The term “attacker” or “attackers” is used interchangeably throughout this report with malware authors, malware operators, and is taken to mean the people using of writing financial malware.
Packed

Malware samples are often “packed”. Packing a sample means encrypting or obfuscating its contents to hide the nature of its functionality. Typically an unpacking stub is inserted into the file which unpacks the sample when it is executed.

Yara

A tool and language used for pattern matching

1.4 Malware Families

This small section gives a very brief introduction to the malware families mentioned in this report and, where applicable, references for further information.

*Trojan.Blinder:* Early Trojan using JS to overlay location bar in browser  

*PWSteal.Goldun.B:* BHO Trojan that steal account credentials for e-gold service

*Trojan.PWS.ChromeInject.A:* Malicious Firefox extension

*Win32/Theola.F:* Malicious Chrome extension

*Mebroot:* Kernel-rootkit banking malware family

*Zeus:* Infamous banking malware family

*Citadel:* Advanced variant of Zeus based on leaked source code

*Gameover Zeus:* Widespread, peer-to-peer based variant of Zeus

*Pony:* Downloader and password stealer

*Gumblar:* Server-side code injector

*Gataka:* MitB family that uses a local proxy

*Gozzi:* Banking malware dating from 2007

*Shylock:* Banking malware that runs over HTTPS and heavily targets the UK

*Ramnit:* Worm and file infector with credential stealing capabilities

*Dexter:* Point of Sale device targeting malware

*Kelihos:* Spambot using a peer-to-peer network
2 Financially Motivated Malware

In this section we give a history of financially motivated malware on Windows desktop computers. We describe that history of terms of the techniques used by in-the-wild malware families to achieve their goals.

There are many facets to a malicious software sample of this nature so we break the techniques down into distinct areas. We begin by examining the various methods used to capture data of interest on an infected system; we then examine the ways in which that data is exfiltrated to attacker-controlled resources, before exploring the techniques used to stealth the presence of the malware.

We then move on to examine how modifications can be made by the attacker to illicit extra information being stolen, and how automation and modularisation have been used to industrialise information theft, then finally exploring techniques used to defeat countermeasures used against financial malware.

2.1 Data Capture

The primary goal of financial malware is to steal money, and to steal money it must steal data. To trigger the whole process the data must first be captured.

Malware has evolved techniques to capture data at a wide range of different points along its path from user to financial institution. This includes when the data is at rest on an infected computer, when the data is entered by the user and when the data is in flight from infected machine to banking website.
Not only has malware evolved to capture data at more points along its journey, but we have also seen an increase in the levels of sophistication employed, with the aims of ensuring the most useful data is captured and is done so more reliably.

2.1.1 Keystroke Loggers

One of the oldest and most widely used approaches is that of keystroke logging. This is the process of recording every key that a user presses on their keyboard. Computer programs that carry out keystroke logging are referred to as keyloggers.

Keyloggers are not inherently malicious. For example a concerned parent may use a keylogger to monitor the online activities of their children, or a business may decide to monitor an employ who they have reason to believe has been carrying out illegal activity using their desktop computer at work.

For these reasons and more there are a variety of commercial software and hardware-based keyloggers available on the market. Keylogging in general is a very active area of continuous research, particularly novel keystroke logging methods such as keyboard acoustics [ZZT05], and motion detection on smartphone devices [CC11].

However, from the perspective of a financial malware author keystroke logging is an obvious way to capture usernames and passwords and various other pieces of data that the user types into their keyboard. Hence, keystroke loggers capture data when it is entered by the victim.

In this section we examine the two most common methods used by financial malware on Windows desktop computers to capture keystrokes.

2.1.1.1 SetWindowsHook

SetWindowsHook and SetWindowsHookEx are Windows Application Programming Interfaces (APIs) that, according to MSDN [MSDN00]:

“Installs an application-defined hook procedure into a hook chain. You would install a hook procedure to monitor the system for certain types of events. These events are associated either with a specific thread or with all threads in the same desktop as the calling thread.”

These APIs are used by malware to setup a routine that will be called for every key press event, thus the routine can log every key entered by the user. Calling SetWindowsHookEx with the idHook parameter set to WH_KEYBOARD will install a hook procedure that monitors keystroke presses. This technique has been documented in financial malware at least as early as 2005 in [HS05].


2.1.2 Window-based APIs

The second common method of keystroke logging is through using a slightly different suite of Windows APIs that centre around the operating system’s use of windows (with a small ‘w’) to represent elements of the desktop experience.

A software program can use a combination of window-based APIs to establish which key may have been pressed and by running in a loop can record a continuous sequence of key presses. As documented in [JC05], malware uses GetAsyncKeyState to establish whether a key is up or down.

A particular benefit of this approach over SetWindowsHook is that other window-based APIs can be used to reduce the applications for which keystrokes are recorded. If the malware is only interested in logging keystrokes from a browser process or banking application then it can use other APIs such as GetForegroundWindow can be used to find the name of the application that is currently in focus. GetWindowText can be used to establish if the window has changed and only then will keystroke logging start.

Using the targeted approach the malware operators receiving the stolen data will have higher proportion of genuinely useful data captured. Whereas a non-discriminatory keylogger will capture everything and the operator will have to sift through the entire log, including emails, typed internet addresses, typed commands and all kinds of other uninteresting data to retrieve the usernames and passwords that are of the highest value.

2.1.3 Interface Replacement

Interface replacement is a technique used by some banking malware that overlays elements or the entirety of the genuine interface to the banking portal with an interface that belongs to the malware.

The goals can be to conceal areas of the interface that may alert the user that their session is being tampered with or, in terms of data capture, to intercept the information entered by the user and redirect it to an attacker-owned resource such as a remote website. This technique captures data as it is entered by the victim.

2.1.3.1 Website

A web interface is the most common medium through which people interact with their bank accounts online. It therefore comes as no surprise that the interface replacement technique has been used against banking websites by overlaying the web browser window itself.
Some early examples of areas of the website being overlaid include Trojan.Blinder documented in [CW05] that used JavaScript to place a white box over the location bar in the browser that contained the address of a legitimate banking website, when in fact the user was visiting a phishing page. Thus the technique is used to considerably enhance the believability of a phishing scam.

Other tactics include hiding warning messages with blank pieces of background, and overlaying a different form field over username and password boxes that will intercept the details of the user as they are entered.

2.1.3.2 Banking Application

Many banks, particularly in certain regions of the world such as South America, issue a standalone banking application for their customers to use to interact with their bank account.

A similar technique to website overlay is used to overlay the interface of the banking application. The unsuspecting user then enters their banking details into the fake interface and the data is sent to the attacker.

2.1.4 Man-in-the-Browser

Man-in-the-Browser (MitB) is a technique whereby malicious code runs from inside the browser process, and therefore has access to all the information that the browser has access to, with particular focus on login credentials and other details exposed during a victim’s online banking session.

Again, it is worth emphasising that the web browser is the primary means through which users interact with their bank account online. We have already discussed how the browser is targeted through interface replacement in [2.1.3.1], MitB is the natural evolution in techniques that target the browser.

By executing code directly from within the browser, the malware can carry out a huge range of tasks including silently monitoring the data that the user enters, modifying the information that gets presented to the user, intercepting and denying certain URL requests, bypassing encryption mechanisms such as HTTPS and TLS by accessing data before it is encrypted, and many more.

This technique allows malware to capture data when it is both entered by the victim and as it is flight from browser to the internet.
Browser Extensions

2.1.4.1 Browser Extensions

Browser extensions are mechanisms by which software developers can add functionality to or “extend” the behaviour of the web browser. Most of the major browsers have a browser extension interface and there are many legitimate and popular examples, including extensions for defensive purposes such as disabling scripts or to block adverts, and extensions that add translating functionality.

Browser extensions are loaded into the browser process and they therefore have a certain degree of access to the data in the browsing session of the user. This access can include usernames and passwords and other data of interest to cyber criminals. This access makes the browser extension a popular delivery mechanism for malicious code.

Three of the most common types of malicious browser extension are Browser Helper Objects, Firefox Extensions and Chrome Extensions.

Browser Helper Objects (BHO’s) are DLL modules that are used to extend the functionality of Microsoft’s web browser, Internet Explorer. They were introduced in version 4 and had unfettered access to the Document Object Model (DOM) of the browser. This meant they could access all areas of the browsing session including all usernames and passwords entered by the user, thus there have been many examples of malicious BHO’s designed for financial gain.

A malicious BHO is loaded into the address space of the browser and can monitor URLs visited and record keystrokes entered when the user logs on to their online banking website. Since the BHO also has access to the functionality of the whole browser it can also send the stolen information out to attacker-controlled addresses from within the browser process. Thus, a BHO is the only piece of malicious code needed to have a fully-functional banking Trojan.

As an example we present PWSteal.Goldun.B, documented in [CW05]. BHO’s were a popular delivery vector for malware for earlier versions of Internet Explorer. More recent releases considerably isolated the access given to BHO’s resulting in fewer malicious examples.

The Firefox browser has also had the capability to be extended for many years. Although the interface for extensions to use, and therefore the specific makeup of the extension itself, has changed through Firefox versions, early incarnations had similar levels of access to the browsing session of the user as Internet Explorer extensions.

An example of a malicious Firefox extension that targets online banking is Trojan.PWS.ChromeInject.A outlined in [VC08].

The Chrome browser from Google is another popular browser that can have its functionality extended through third party modules. Chrome extensions have a similar makeup to Firefox extensions in that they are usually an archive containing a manifest file and several other files that hold the functionality of the extension.
Malicious Chrome extensions have been used as part of larger financial malware suites such as the case of *Win32/Theola.F* which was used by *Mebroot* banking malware family as documented in [AM13].

### 2.1.4.1 Process Injection

It is possible for malware to control the browser without using any interface provided by the browser developer. The first key step in a generic MitB attack is for the malware to have its code executing in the context of the browser process, this is known as process injection.

The most typical method used is to open a handle to the browser process, allocate memory inside the process, write the code and data for the malicious program into the newly allocated memory, and then start a new thread of execution inside the malicious code that carries out the malware payload. [Figure 1] shows the process injection sequence.

[Figure 1 - process injection sequence]

This is a sequence of events that can easily be carried out using Windows APIs. To get a handle to the target process we can use `OpenProcess`, to allocate memory inside the target process we can use `VirtualAllocEx`, to write code into the address space of the target process we can use...
WriteProcessMemory, and then finally a thread of execution can be started inside the newly injected malicious code using CreateRemoteThread. More information on these APIs can be found in the Microsoft Developer Network documentation.

A slight variation on this theme is to use the above method to load a DLL into the target process. The steps are identical with the exception that the code written into the target process is merely and very short snippet that contains a call to the LoadLibrary API with a string parameter that contains the path to the payload DLL on the disk.

The DLL loading approach has the advantage that the payload data can be compiled as a normal DLL that the Windows OS will load correctly including resolving the import table and other tasks that the OS normally carries out. However, loaded DLLs can easily be enumerated using standard OS tools such as Task Manager. If the full malicious payload is loaded into the target process and no DLL is loaded then the initial thread must carry out tasks that the Windows loader would otherwise complete. The benefit to the attacker is that the malicious code is less easy to detect as there is no giveaway loaded module in the loaded module list. [figure 2] shows the loaded modules list as displayed using the SysInternals tool Process Explorer.

[figure 2 - loaded dlls]

The former variation of the technique is also not immune from detection, both after the fact and as the injection is taking place.

The presence of the injected code can be found after the fact through several means. One way is to enumerate the threads currently executing in the process. Threads have associated with them an address that corresponds to the location in memory from which the thread was initiated, called the thread start address. Legitimate threads will have a thread start address located inside the main module of the process or one of the DLL modules contained in the loaded modules list. A thread running from injected code will have a thread start address that is not located in any loaded module.

Injected blocks of code can also be detected based on the characteristics of the memory pages they are located in, particularly the permissions which are typically READ, WRITE and EXECUTE. Using these kinds of heuristics potentially injected code can be found and examined. There
exist several tools to automate this process including the Volatility [3.3.3.1] plugin malfind described in [ML11].

Since the API sequence outlined above is highly indicative of process injection many host-based security solutions, such as endpoint Antivirus, treat it as suspicious and may block the new thread from being started in the target process.

For this reason there are a variety of other injection methods that have been developed, however their details are beyond the scope of this report. Further details can be found in [ML11-pp.596] and [AM12].

2.1.4.2 API Hooking

One of the main functions that MitB malware will carry out once it has been injected into the browser is to perform API hooking.

API hooking is the process of subverting the control flow of Windows APIs in a particular process so that the hooking code can intercept the arguments passed to the API and perform some action on that data.

An obvious example is to hook the send function found in the ws2_32.dll library. This function is part of the Windows implementation of UNIX sockets and is responsible for sending data on a connected socket. Malware may hook this function in order to examine the data that is sent by the browser using this API and sift it for any login credentials.

To hook an API the malicious code must divert the control flow away from the legitimate path that normal execution would take, to an alternate path dictated by the malware. This can be achieved in a number of ways.

The first technique is called Import Address Table Hooking. The Import Address Table (IAT) is a structure inside Windows Portable Executable (PE) files that lists the APIs that the program uses during execution. When Windows loads the PE file into memory it adds the address of each API into the IAT. For more details on the exact nature of the IAT see [MP02].

To hook the IAT, the malware must find the relevant structure in its process memory and then substitute the address of the genuine API with an address that points into malware code. For a detailed description of IAT hooking see [BB09a].

A second technique is called detour patching, inline hooking, or trampoline hooking. Rather than altering the call table as IAT hooking does, detour patching overwrites the actual bytes of code at the function address with alternative code that performs the tasks required by the malware. Typically the initial instructions will be a jump to a different address in memory, called a trampoline. This reduces the amount of instructions that need to be overwritten at the function address, which reduces the likelihood that errors are introduced which might cause the compromised process to crash.
Although more complex, this approach is very flexible and can result in hooks that are harder to detect than IAT hooks. This technique has been developed by Microsoft into a commercial product called *Detours* which was first proposed in [HB99], details can also be found in [BB09b].

Once an API has been hooked malware writers can be very creative in how they use that hook. For example the *HTTPSendRequest* API may be hooked so that any data sent in a HTTP request can be monitored. This may include POST data that contains usernames and passwords, or the URL being visited so that the malware can block the request or alter the data that is received back [2.4.1].

API hooking can be used to facilitate many other types of data capture and modification that we discuss in other sections of this report such as screen capture [2.1.5], clipboard capture [2.1.6], and DNS redirection [2.1.9].

2.1.5 Screen Capture

Screen capture is the process of taking a record of the exact contents of the screen of the victim PC. It became a necessary part of financial malware’s arsenal after some banks attempted to introduce mechanisms to thwart simple username and password theft that financial malware was successfully using, such as virtual keyboards that the user must type on using the mouse or selecting elements of a memorable data word from a drop-down menu. These devices meant that malware that simply recorded keystrokes would not capture the data that the infected user entered.

To counter this countermeasure the malware authors started to introduce screen capture functionality into their software.

This type of data capture can be said to take place on data as it is entered by the victim, as the data being captured is the direct actions of the victim.

2.1.5.1 Screenshots

A *screenshot* is a still capture of the visual state of the PC at a particular moment in time. It is usually taken as an image file and sent back to the attacker.

The attacker may be interested in obtaining a screenshot when the victim accesses a website that requires the user use something other than their keyboard to login, such as virtual keyboard or a drop-down menu.

A screenshot is generally taken when the malware receives a command from its control server or more likely when a specific event occurs as discussed in [2.1.5.2].

Financial malware that is capable of taking screenshots has been discussed as early as [CW05].
2.1.5.2 Video

Video capture is an extension of screen capture with the obvious exception that the capture is now over a period of time and will take the form of a video rather than image file.

Video rather than still image became a viable option as the average bandwidth available to personal computers on the internet increased to such an extent that uploading a video from the infected machine would not take so long as to be unworkable.

When there are multiple operations that need to be captured such as multiple mouse clicks then digesting a single video capture may be a simpler and quicker process than digesting a long sequence of still images. Video capture can also be used by the attacker to gain a better understanding of how an online banking application functions and how the user interacts with it, information that could be used to craft new features for the banking malware.

An example of malware that includes video recording functionality is Citadel described in [JM12].

2.1.5.3 Event-driven

Capturing random screenshots or snippets of video will produce huge amounts of irrelevant data that must be manually processed by the malware controller. This is obviously a huge waste of time and renders the screen capture functionality almost useless. However, if the screen capture occurs at specific points then the noise can be vastly reduced and the malware controller need only look at useful data.

Malware authors have implemented event driven screen capture in a number of ways. The most popular method is to have a section in the configuration file used by the malware that lists the URLs that a screen capture must take place for. Typically, the malware will check the current URL being visited against this list and initiate a video capture or will take a screenshot when the mouse is clicked.

2.1.6 Clipboard Capture

Capturing data that has been copied to the clipboard is a novel approach that several banking malware families added to their repertoire once it was realised that valuable data may sometimes only be copied and pasted and not typed.

This is particularly true of data such as passwords. There exist many password management tools that are designed to make it easy for users to have multiple different complex passwords for their online accounts, but not have to undertake the onerous task having to memorise every one of
them. Password management tools help by storing all the account details in an encrypted file, protected with one master password. The user now only has to remember one password rather than a whole range.

Such password management tools frequently offer a secondary service of generating random passwords that the user can apply to a certain account. In these cases in particular the user is very likely to simply copy the password and paste it into the password box of the login form.

Malware usually captures clipboard data by hooking the GetClipboardData API, as described in the analysis of Zeus in [ML06].

Capturing data from the clipboard could be said to be capturing data as it is entered, but we can also say that it captures data at rest to a certain extent since data that is already on the clipboard can be captured.

2.1.7 Stored Credential Theft

In the name of convenience, many varieties of software on desktop PC's that require some sort of authentication offer a facility to store the credentials so that they do not need to be remembered and typed in every time the user wishes to use the software and log in to access a particular feature. Typically, the software will save the credentials for the user and automatically apply them when the user next wishes to log in.

From the perspective of banking malware, storing credentials in this way represents a potential avenue to bypass their information stealing capabilities. The user does not enter any information, as it is already saved, so there is no data for the malware to capture.

However, these applications must store the saved credentials in such a way that they are recoverable and can be replayed by the software when a new login is required. In many cases, those credentials are stored in a manner that allows malicious code running on the PC to also recover them.

So, rather than a hindrance to data capture, stored credentials become an opportunity to steal more username and password combinations than might otherwise have been witnessed by the malware, and to steal them immediately that the PC is infected.

Capturing data using this technique steals data when it is at rest on the infected computer.

2.1.7.1 Browsers
The browser is used to log in to all manner of websites, including banking websites, and so it is natural that all major browsers offer some sort of ability to save credentials for individual websites.

In many cases the browser also has a facility to reveal the saved passwords to the user, which is useful if the user forgets the password and needs to use it again elsewhere. This means that the passwords are stored in a reversible way which can be replicated by malware.

Revealing saved credentials in this way is so common that there exist third party tools to perform the act such as IE PassView from NirSoft. Internet Explorer has used a mechanism called Protected Storage or PSTORE in the past to save credentials. This mechanism is not secure against third party programs and has been abused by malware including Zeus as documented in [NF09].

2.1.7.2 Email Clients

Email clients are another class of programs that financial malware frequently targets for stored credential theft. Users are even more likely to wish to have their credentials saved for this type of software than the browser, as email is checked many times per day and re-entering the username and password every time would be extremely cumbersome.

Although email account credentials are not directly useful to financial malware, users are often lazy and re-use the same passwords for multiple accounts. They may use their email account password for their online bank account.

The email account credentials themselves can also be useful in their own right. They can be sold or they could be used to spread the financial malware. The stolen credentials may be used to send an email directly from the infected user’s account, with a copy of the malware attached.

The principles that apply to web browser saved credentials apply to email clients as well, namely, they must be stored in a reversible way. Again, third party applications, including malicious applications, have the capability to reverse the process and retrieve the plaintext credentials.

Applications exist to perform password recovery from all popular email clients such as Mail Passview from Nirsoft, and a great many malware families have this capability such as Pony. [AD13] details a Pony botnet whose database of stolen data, including huge numbers of email client passwords, was uncovered by security researchers.

2.1.7.3 FTP Clients

FTP clients are another class of programs that frequently offer an option to save the user’s login credentials. There are a wide variety of FTP client programs available and some malware families have the ability to steal the credentials from a very large number of them. [AD13] also details FTP client credential theft.
An interesting aspect of theft of this type of credential is that the primary goal is normally to further spread the infection of the malware family. It is very common for the FTP servers that the client programs are used to upload files to, also house a web server. In fact, FTP clients are often used to upload files that will appear on a website.

If malware authors manage to obtain credentials used to upload these files then there is a good chance that they are now able to upload arbitrary files to web servers on the internet. This has been used as a way to implant malicious server-side code onto websites that will redirect anyone visiting the website to download and install malware onto the victim’s machine, possibly exploiting browser vulnerabilities to deliver the malware without the victim’s knowledge.

There was a high profile instance of this activity in 2009 when the Gumblar family inserted malicious code into thousands of legitimate sites using compromised FTP credentials, as documented in [BP09].

### 2.1.7.4 Cookies

Cookies are a form of stored credential that achieves the result of the client logging in without having to re-enter their credentials. Cookies differ from other types of stored credentials as the credentials themselves are not actually stored, but rather a small file which acts as a unique identifier for the user’s session.

Rather than having the user enter their password every time they visit a website, a cookie can be saved on their hard disk and used instead. In this case, the cookie files effectively become authentication tokens, stealing them may allow an attacker to impersonate the genuine owner of the cookie and login to their account.

In addition to simple theft of the cookies, many malware families also have an option to delete cookie files that are found on the victim machine. Since cookies mean that the user does not need to re-supply them login details when they next visit a website, any malware on the machine will not capture those credentials. Deleting the cookies ensures that the user will need to enter their credentials next time around, thus allowing the malware to capture them as they are entered.

### 2.1.8 Form Grabbing

Form grabbing is name often given to the process of capturing data that has been entered into web forms, typically usernames, passwords and other credentials. This data has a very low noise ratio compared to keylogging, meaning that the majority of the data captured in this way will be high value.
Generic approaches to form grabbing implemented by malware use MitB techniques outlined in [2.1.4] and hook APIs inside the browser process responsible for crafting the HTTP data and sending it to the web server. Typically, HTTPSendRequest is hooked which delivers the complete HTTP data to the server. When a form submission is made from a web page it will normally be sent a POST request. The hook code inside HTTPSendRequest will check to see if this is a POST request and if so, it will decompose the parameters and send the information back to the attacker. More details of this technique can be found in [AS11c].

Capturing all data that the victim enters into forms has reduced the noise from blanket keylogging, but there is still large amounts of extraneous data that must be waded through by the attacker to find the most useful items. Forms are not just used to enter credentials but are also used for a variety of other functions on web pages. Additionally, malware botnets can sometimes harvest so many credentials that it becomes a sizeable task to filter out the most prized accounts from the relative dross. This decreases the overall value of the captured data as there is time and work required to sieve the wheat from the chaff.

This has led to many financial malware families featuring selective form grabbing. The list of websites for which form grabbing is enabled is dynamically controlled in the configuration file that many malware families distribute to infected machines. This is a similar type of construct as the selective keystroke logging described in [2.1.1]. The list of websites may be a flat list but is more likely to be a list of regular expressions or similar variable pattern that is compared to the current URL.

We can say that form grabbing captures data as it is entered by the user but also captures it in flight to a certain extent, as some data that the user did not enter will be captured when the form is submitted.

2.1.9 Local Proxy

This technique captures data by redirecting all network traffic from a particular process to another local process or more likely a module injected into the process whose traffic is being hijacked. The proxy module can then extract any information it wishes from the data before relaying it to a remote address. The proxy module can also choose to modify the data sent or received [2.4] or reject the traffic outright.

This is perhaps a “cleaner” or certainly more simple, approach than traditional MitB techniques. When it comes to simple data capture the simplicity is less evident, but when we look into data modification in [2.4] then it will become apparent. Using a proxy to inspect, filter and act on all the network traffic requires relatively few API hooks, with all the main functional logic contained within the proxy module. In contrast, a MitB approach that does not use a local proxy must hook many APIs that deal with crafting, sending, receiving and adjusting network traffic, particularly higher level APIs. The data that passes between these APIs must be managed by the MitB code which can add complexity and potentially introduce errors. Hooking lower level APIs and
using the proxy approach may also be more browser-agnostic meaning the interception code can be ported to different browsers with fewer modifications.

One problem that the proxy approach introduces is that data sent using HTTPS may already be encrypted, whereas the higher level API hooking approach will see the data before it is encrypted. This problem has been surmounted by malware authors through a man-in-the-middle (MITM) attack.

In this attack the malware intercepts the client’s attempt to establish a secure connection with a remote server, and impersonates the remote server by sending its own public key and certificate which the client then uses to subsequently encrypt the data. Meanwhile, the malware establishes its own secure connection with the remote server and relays the server’s responses back to the client, re-encrypting them with its own key in between. The client believes it is communicating with the remote server over a secure channel when in fact the malware is intercepting and relaying all traffic.

For this strategy to work, the client must believe that the public key being offered by the malware genuinely belongs to the banking website that the client is trying to communicate. This means it must be signed with a certificate whose certificate chain the browser ultimately trusts. In theory, this should not be possible as the malware authors do not have access to the private signing key of the banking website in order to sign the public key belonging to the malware. However, since the malware has unrestricted access to the client machine it can subvert the signature verification process in a number of ways.

One method is to install a new trusted root certificate authority on the machine. This can then be used to sign the public key presented by the malware and the signature will be accepted by the client software. A second method is to hook the functions used to verify the signature and change their behaviour so that they incorrectly pass the malware created signature as valid.

A description of the Gataka family using the second of these techniques can be found in [JB12a] and [JB12b].

This data capture technique can be said to capture data in flight since the data is intercepted after it has left the victim’s browser.

2.1.10 Techniques Comparison

We will now summarise the techniques used to capture data from victim machines and rate them based on the following criteria: where data is captured, the effectiveness of the technique at capturing the desired data, the noise level, the difficulty of implementation, its level of stealth, and how prevalent the technique is in the wild.


<table>
<thead>
<tr>
<th>Technique</th>
<th>Where</th>
<th>Effectiveness</th>
<th>Noise</th>
<th>Difficulty</th>
<th>Stealth</th>
<th>Prevalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keystroke logging</td>
<td>Entered</td>
<td>Medium</td>
<td>High (can be reduced with selectivity)</td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Interface replacement</td>
<td>Entered</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>MitB</td>
<td>Entered/inflight</td>
<td>High</td>
<td>Low (can be finely tuned)</td>
<td>Medium</td>
<td>Medium (stealth can be increased with creative hooking)</td>
<td>High</td>
</tr>
<tr>
<td>Screen capture</td>
<td>Entered</td>
<td>Low (only effective in edge cases)</td>
<td>Low</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Clipboard capture</td>
<td>Entered/rest</td>
<td>Low</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Stored credential theft</td>
<td>Rest</td>
<td>Medium</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Form grabbing</td>
<td>Entered/inflight</td>
<td>High</td>
<td>Low</td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Local proxy</td>
<td>Inflight</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>High</td>
<td>Low</td>
</tr>
</tbody>
</table>

(Table 1 – data capture techniques comparison)

(Table 1) shows that despite certain techniques being relatively ineffective, such as screen capture which if used in isolation which capture only a small amount of data, they can be highly prevalent as they are typically just one technique used by a financial malware family. Many of the more difficult techniques such as MitB and form grabbing are highly prevalent as the code to accomplish the technique has been widely shared and publicised, resulting in re-use.

2.2 Data Exfiltration

Once malware has captured the data that it is interested in it must transfer that data in some way back to the malware controllers. Data exfiltration is a fundamental requirement of any financial malware as stolen data that the attackers do not receive is useless data.

This section describes the various techniques that financial malware has used to satisfy its requirement to exfiltrate data. Although all of these techniques use the internet as the delivery
mechanism, there have been documented cases of malware exfiltrating data from air-gapped networks using techniques such as copying to removable drives, though an analysis of those techniques are beyond the scope of this report. An example of data exfiltration from air-gapped networks can be found in [JC14].

2.2.1 HTTP

HTTP is the language of the World Wide Web and as such there is a huge amount of HTTP traffic passing across the internet at any point in time. It is a critical protocol in most environments meaning that it is very likely to be allowed through firewalls.

As a result HTTP is very commonly used by financial malware to exfiltrate data. The malicious traffic can blend in with the genuine traffic, it is very likely to make it through network security measures as HTTP in general must be allowed, and it is highly versatile allowing all kinds of different data types to be transferred.

2.2.1.1 Plain Text

Some of the earliest examples of malware using HTTP to exfiltrate stolen data simply sent the data back to a command and control server in plain text. The stolen data may include usernames, passwords, credit card numbers, bank account details and other highly sensitive material. By sending this data across the internet in unencrypted form the malware is exposing the victim to even greater risk (if that is possible), as it may be accessed and used by parties other than the malware operator.

[CW05] highlights the case of PWSteal.Goldun.B which was discovered to have been send data in plaintext using a HTTP POST request. The stolen data was then stored in a plaintext file on the server and was globally accessible to anyone who browsed to the correct URL.

[DJ07] describes the case of Gozi discovered in 2007 skimming all POST requests sent by the browser on the infecting machine, including any sent over secure HTTPS, and sending them via insecure HTTP to the attacker’s command and control server.

The upside to plain text exfiltration from a network defender’s point of view is that the outbound traffic can be scanned and rules applied to attempt to identify the types of data that financial malware typically attempt to steal. Network security devices or Data Leakage Prevention (DLP) tools may then be able to block that data from leaving the network.

2.2.1.2 Weak Encryption
The next logical step from plain text exfiltration of data over HTTP is to obfuscate the data in some way. This can be a simple transformation that is trivial to reverse but nevertheless serves the purpose of disguising the data as meaningless garbage to the casual observer, or automated DLP.

Simple single or multi-byte XOR based encryption is a popular technique, as are similar schemes around simple logical operations on bits or bytes of the plaintext. In many cases there will be some additional encoding of the data on top such as base 64.

2.2.1.3 Symmetric Key Encryption

The next advance in HTTP data exfiltration is to use robust symmetric key encryption. This ensures that the data cannot be read without access to the key, ensuring that any third party observing the communications between the infected machine and the malware command and control server cannot read the data passing between them. Not only does this include the exfiltrated data but also any commands or other information that passes between client and server.

There have been several approaches used to establish the encrypted communications channel. In some cases a key is embedded into the malware executable and pre-shared with the server component. This offers a reasonably high level of security. As long as the encryption is implemented without errors, the communications between infected endpoint and controlling server should be unreadable without access to the key, which must either be retrieved from the server or extracted from the binary on the infected machine. Access to the server is unlikely unless it has been seized by law enforcement and extracting the key from the malware executable is a non-trivial task that must involve reverse engineering the malware and extracting the key.

Variations on this theme exist. For example, there may be one fixed key for a whole botnet, or each individual sample may have its own unique key. The former strategy simplifies the administration of the botnet. The server can decrypt all incoming traffic with the same key and each new malicious executable file can be repackaged and distributed without having to change the embedded key.

However, this approach implies that when the single encryption key is discovered through reverse engineering the malware sample, all communications past, present and future between all bots and the server can be decrypted. The second approach of having a unique key for every unique executable can quickly become a logistical problem. Malware authors often re-package binaries producing tens or hundreds of thousands of unique files. If each file has its own key the server must store and keep track of a huge number of different keys. In practice, malware authors tend to use a hybrid approach that compromises a slightly lower level of security with a degree of ease of management.

Zeus is an example of a family that embeds a decryption key into the malware executable. Keys are constant for each distinct botnet but a botnet master may choose to divide their botnet up into many botnets each with their own keys. [JW11] describes the use of RC4 as symmetric encryption between client and server in Zeus. [JM12] describes the use of AES for communications and data exfiltration as used by Citadel.
2.2.1.4 Public Key Encryption

The issue with symmetric key cryptography is that there must exist a key that is shared between the client and the server. If this is discovered by a third party then communications can be decoded. On the other hand, if used in the correct way, public key cryptography has the potential to render communications between the client malware and its control server completely unreadable to third parties.

Malware that uses an embedded public key to encrypt all communications data before sending it back to its control server would pose a serious challenge to anyone attempting to read the network data. However, banking malware communicates with its command and control servers frequently and can exfiltrate very large amounts of data. This requires data to be encrypted a very large number of times, and since public key cryptography is generally very slow to encrypt data in comparison to symmetric key cryptography, this kind of pure public key based encryption has not been seen in the wild.

However there are many examples of malware using a combination of public and symmetric key cryptography during the process of data exfiltration. A common example is for elements of the exchange between client and server to be signed and verified using asymmetric algorithms. Updates to the main malware module, plugin files and configuration data are frequently signed to prevent the data from being hijacked on route and hinder takeover of the botnet by law enforcement.

There are also cases where symmetric key encryption is used in conjunction with public key encryption. An example is when a session key is generated by the client that will be used in a symmetric algorithm such as AES and it is encrypted with a public key and sent back to the command and control server. The session key is then used to encrypt subsequent communications between client and server. [SV14] describes the case of a family known as Gameover Zeus encrypting an AES256 session key with an RSA public key embedded in the client executable.

Encrypting communications in this way makes it very difficult for a third party to read the data. The third party must typically have access to the endpoint while the malware is executing and retrieve the session key from memory.

2.2.2 HTTPS

Exfiltrating data over HTTPS is a simple way of rendering the data unintelligible to the casual observer. Indeed, HTTPS can render the data unintelligible to even the most hard working third party observer.

Furthermore, HTTPS does not require any implementation considerations for the client executable. The Windows APIs used to send HTTP data will also send data over HTTPS with very few
modifications. HTTPS would therefore appear to be an excellent way for a malware client to communicate with its command and control server over a secure channel.

However, there are drawbacks to using HTTPS. The first is that it can be decoded using a man-in-the-middle attack, and many security gateway products offer this capability. Secondly, the attacker’s command and control server must have a certificate which is transferred in plaintext in the network traffic during the certificate exchange. This can act as a means to detect and block the traffic associated with this malware family.

There are several cases of financial malware families that exclusively use HTTPS for data exfiltration. *Shylock* as described in [DE13] and *Ramnit* described in [GT11] are two examples.

### 2.2.3 FTP

FTP is a protocol commonly used to transfer large files between hosts. This would make it a robust option for financial malware to exfiltrate large amounts of data.

However, for a variety of reasons, FTP is a relatively uncommon choice for data exfiltration amongst the types of financial malware within the scope of this report. The main reason for this is that it is much simpler for malware to send stolen data back over the same channel as other command and control data is sent. FTP does not fare well as a command and control channel as it is primarily designed to transfer large files, and short bursts of information common to control channels do not carry well. If FTP is used to exfiltrate data and an alternative protocol such as HTTP is used for command and control there is the added complexity needed of coordinating between the FTP server and the HTTP server. If all data is going to the HTTP server this complexity is eliminated.

FTP tends to feature as a data exfiltration mechanism more commonly in certain types of malware where the quantity of data being stolen is so vast that FTP becomes more attractive than the alternatives. When the data being stolen is all of the same type for example track one and track two data from payment cards, the case for FTP becomes stronger. This is evidenced by malware families that steal payment card data from Point of Sale (PoS) devices, such as *Dexter* described in [CW13].

### 2.2.4 SMTP

SMTP is a protocol used to send email messages. When financial malware exfiltrates data over SMTP it is generally emailing it to an address under the control of the attacker.

This is a relatively simple way to get stolen data off the victim machine that allows for easy retrieval by simply accessing the target email account. However, it does not scale well as frequently emailing large amounts of data is not always practical, and it exposes the data to interception or blocking as it passes through the systems of the email provider. On the plus side the attacker does
not need to set up their own infrastructure to receive the stolen data and small amounts of traffic may blend in easily with normal network activity.

SMTP exfiltration is still seen in certain types of malware such as Remote Access Tools, traditional keyloggers and password stealers and is used by some advanced threat actors such as those described in *Operation Cleaver* [CY14].

### 2.2.5 Custom Protocols

One way to add obfuscation to the network communications of malware is to implement a custom protocol. The custom protocol can be entirely standalone, operating on its own port numbers, or more commonly may be layered on top of another protocol such as HTTP.

The obvious drawback is that there is a considerable development cost in designing and implementing a custom protocol. A degree of skill is also required to ensure that the protocol is reliable and offers the level of security required.

The complexity of custom protocols varies considerably. Many malware families that exfiltrate data over HTTP [2.2.1] will also implement a custom protocol on top to obfuscate the data to implement the necessary command and control mechanisms such as issuing commands to the infected machine. These layered protocols are not typically very complicated and are mostly concerned with encrypting and structuring the data. However, custom protocols that are implemented from the transport layer can become very complex.

There are considerable advantages to using a custom protocol. It takes much time and effort for researchers to reverse engineer a new protocol, so details of the network traffic are more likely to stay hidden for longer. Designing a custom protocol from the ground up also means that it can be more focused on the needs of the malware. It is not restricted by the limitations of an existing protocol so advanced features such as peer-to-peer [2.2.6] abilities, encryption, authentication and non-repudiation can be implemented if required. *Gameover Zeus* is an example of a very successful malware family that used its own custom protocol. [CP13] explains this protocol in great detail.

### 2.2.6 Peer-to-Peer

There are inherent problems with the traditional client/server model used by most malware, and indeed, by most services that function over the internet. The main problem is that the server acts a single point of failure. Attacking the server by taking it down or assuming control of it renders all the infected clients directionless. A solution to this single point of failure is to use a peer-to-peer model for the network.
In a peer-to-peer network each node acts as both client and server. Therefore, if one node is taken out then the rest of the network carries on as normal. This approach is inherently more resilient to takedown than the traditional client server model.

Although there are clear advantages to a peer-to-peer network for data exfiltration there are also drawbacks. Designing a network from the ground up is a very complex task and mistakes can be made which render the network vulnerable to hijacking. Examples where this has happened in the real world are the Kelihos botnet described in [TW11], and the ZeroAccess botnet [AN13] which have both suffered from takeover due to failings in their respective peer-to-peer protocols. Even if the network itself is not vulnerable to technological take over Operation Tovar [DJ14] which was a coordinated action from worldwide law enforcement against the Gameover botnet shows that no network is invulnerable.

2.2.7 Tunneling

Tunneling in the context of malware data exfiltration is the practice of embedding the traffic inside another protocol. This is similar conceptually to a network tunnelling protocol such as Layer 2 Tunneling Protocol (L2TP), however, malware has demonstrated use of tunneling in an attempt to hide its traffic from observers.

The classic example is tunneling data over DNS. Malware can encode data into the fields of DNS packets and send the data to a DNS server that they control. The subsequent traffic appears to be genuine (if unusual) DNS data that may be overlooked in packet captures or by network based security solutions. [CD11] analyses DNS as a command and control channel.

2.2.8 The Darknet

Darknets is a term used to describe private networks that operate over the internet but are not accessible without using special software specific to the particular darknet in question. They are often created for the purpose of enabling free speech in jurisdictions where the internet is censored and controlled. They frequently feature some kind of anonymous access and many include the ability to set up services that are contained only within the darknet in question.

Malware has been seen to use several of these darknets for elements of their command and control and data exfiltration. The attraction is that in many cases the server that data is sent to cannot be traced back to a physical machine so the likelihood of being caught by law enforcement is very low.

2.2.8.1 Tor
Tor (The onion router) is perhaps the most well-known darknet. Tor includes a feature called *hidden services* which allows members of the network to host services such as web servers and instant messaging servers that can be accessed by other members of the network. Hidden services are designed in such a way that the real IP address of the machine hosting the hidden service cannot be found. [figure 3] shows a diagram of Alice connecting to a hidden service advertised by Bob.

Malware authors use Tor hidden services to host their command and control servers, and typically include a client module or even the genuine Tor software to access the Tor network. [DT13] describes a case of a Zeus variant that uses Tor for its command and control.

### 2.2.8.2 I2P

I2P (Invisible Internet Project) is another example of a darknet that malware has used for command and control and data exfiltration. I2P has a similar feature to Tor hidden services called *eepsites*. These function in a very similar way to hidden services and have been used by malware authors in a similar way, as anonymous command and control servers.

Examples of malware using I2P include *i2Ninja* described in [EM13] and *Dyreza* in [SG14].
2.2.9 Domain Generation Algorithms

Domain Generation Algorithms (DGAs) are a tool used during data exfiltration to generate the domain name that will be used when contacting the command and control server. DGAs are usually designed to deterministically generate a given number of domains over a certain time period. The malware will then attempt to contact each generated domain until it finds an active server.

A DGA essentially means that a malware sample potentially has a large number of different command and control addresses. This makes attacking the malware through server takedown a more difficult task as the malware owner need only register one domain, whereas all potential domains must be neutralised to effectively combat the botnet.

DGAs are usually custom-implemented but commonly use a seed value combined with an element of the current time or date on the infected machine. Conficker is one of the most well-known malware families to employ a DGA, detailed in [PP09], but there are many other examples including Gameover Zeus, see [CP13].

2.2.10 Multiple Fall-back Options

An increasingly common trend in modern financial malware is to have multiple fall-back options for command and control and data exfiltration. There will often be a primary mechanism but if that cannot be reached the malware will try a secondary channel and so on for multiple layers. This adds redundancy if there are failures in the first order options.

Fixed HTTP servers are a common tactic for primary servers, but these can be taken down or blocked by security devices quite easily. A secondary channel may be to employ a DGA and attempt to contact the generated domains. The final layer may then be to attempt to contact an address on a darknet.

[CP13] describes the redundancy employed by Gameover Zeus which used a DGA if the embedded peer addresses for its peer-to-peer network could not be reached. [SG14] describes Dyreza’s use of fixed IP addresses as primary points of contact, followed by an address on the I2P network and a DGA.

2.3 Stealth

In order to conceal the presence of malware on an infected computer a variety of techniques have been developed by malware authors to hide their software from OS and third party tools.
2.3.2 User-mode Rootkits

User-mode rootkits operate in user-mode memory, typically by hooking APIs, and are a common tool used to stealth malware.

The APIs that Windows uses to enumerate files and registry data are the most common places that malware targets. A filter function is employed to check the data passing through the API for artefacts that belong to the malware. For example Shylock uses a user-mode rootkit to hide evidence of its files on disk and the registry entry that will execute it on system start up, as described in [DE13].

2.3.3 Kernel-mode Rootkits

The principle with kernel-mode rootkits is the same as for user-mode rootkits except that they operate in kernel space. As a result they are able to manipulate Operating System structures at a much lower level, making the stealing much harder to detect. Writing a kernel-mode rootkit is more complicated than a user-mode version as code must all be written in kernel mode which is a more specialised area than the general purpose code that suffices for user-mode. The consequences of errors are also higher as a bug in kernel-mode code will cause a system crash.

Microsoft has also introduced various mechanisms in recent iterations of its Operating System to combat malicious kernel-mode code. These include Early Launch Anti Malware (ELAM), Kernel Patch Protection and Kernel Driver Signing. These features have made it harder for malware authors to get their code into the kernel which has reduced the prevalence of kernel-mode rootkits.

Mebroot [KI08] is an example of a kernel-mode rootkit employed by banking malware. Gameover Zeus also employed a kernel-mode rootkit at one point, as described in [JW14a].

2.4 Data Modification

A significant evolutionary step in the history of banking malware happened when families moved on from simple, passive data capture to altering the data that passed between victim and bank so that more information could be elicited. Malware moved from being a passive observer to an active participant, which greatly enhanced the power and the potential of the software.
2.4.1 Web Page Code Injection

Code injection into web pages, commonly known as \textit{web injects} is a feature that financial malware families have been capable of for several years. The first family to really capitalise on the power of \textit{web injects} and make them an essential piece of financial malware’s armoury was \textit{Zeus}.

Web page code injection is a technique whereby extra data is injected into the web page that appears in the victim’s browser, as they are browsing. This is usually achieved using MiTB techniques [2.1.4] where the APIs used to send and receive data are hooked and the data that passes through them is modified according to the attackers needs.

Most malware families decide what code to inject based on a configuration file that is downloaded during execution. This configuration file has a section devoted entirely to code injections.

2.4.1.1 Extra Fields

Code injections have evolved over time as malware authors have become more skilled in deploying them. The earliest (and still common today) form that web injects took was to inject extra fields into forms on banking web sites.

The extra fields are added to force the victim into entering much more information when they log in to the banking web site than they would normally. This extra information is typically ATM pin numbers, social security numbers and secret information used to reset passwords. [figure 4] shows an extra field added asking for a social security number to be added.
Adding extra fields enables theft of more data than would otherwise be possible with passive data capture. However, we can consider it a primitive technique as it easily gives away the presence of the malware. Both to the victim who becomes suspicious as they are being asked for information they would never normally provide in this situation, and also to the bank who can become aware that there is malware on the machine as the extra fields will be included in the form data that is sent to the bank.

2.4.1.2 Remove Warnings

A second common tactic for web injects is to remove warnings that appear on login pages. The problem of banking malware injecting extra code into login pages is well known among banks, so many include warnings on their login pages informing users that the bank will never ask for certain types of information such as PIN numbers to be submitted. Many malware families that inject extra fields such as those will also inject code to hide the warning messages. [SSR14] describes the case of Snifula using web injects to hide a warning message on Japanese banking websites.

2.4.1.3 Dynamic Injection

Modern web injects frequently make use of code constructs that allow the injected code to be more dynamic and flexible. This is often achieved by injecting a piece of code into the web page that itself pulls in a JavaScript file from a remote location. This allows frequent updates to be made
to the remote file without having to push updates down to the infected machines. The exact code can also be tailored based on the country that the infected machine is in or properties such as the browser version.

Web injects are now used to achieve a variety of advanced functionality such as defeating two factor authentication [2.6.1.2.1] and Automatic Transfer Systems [2.5.2]. [JB14] contains an excellent history of the evolution of web injects.

2.4.4 Clipboard Modification

There have been several cases where the novel approach of modifying data that has been pasted to the clipboard has been used to steal money.

[CP14] describes the case of VBlip that was stealing money from Polish bank accounts by monitoring the clipboard for data that looked like a Polish bank account number. The malware would then change the value to a bank account number controlled by the attackers. Any time that a victim copied and posted a bank account number that they were setting up a transfer for, they would inadvertently paste the attacker’s bank account number.

2.4.5 DNS Redirection

DNS redirection is a broad term to encompass the techniques used by financial malware to redirect domain requests for genuine banking websites to attacker controlled addresses. This is similar to a phishing attack but because it is taking place from the victim machine it is much less likely to be picked up by host-based anti-phishing techniques that browsers use to verify a site is genuine.

Hijacking DNS requests makes it appear to the client machine that the website being visited really does belong to the intended organisation. Having an effective fake website in place that is an exact replica of the genuine website is important to perpetuate the ruse. Any information that is entered into the fake website is captured by the attackers.

DNS redirection can also be used to block access to certain websites. It is a common tactic for malware to attempt to prevent an infected machine from accessing security updates or even for the user to access online help if they suspect there may be a problem with their machine. To achieve this, the malware may redirect DNS requests to security-related domains to a null address or commonly to a neutral address such as Google.

Variations of this type of attack have sometime been referred to as pharming, which is an amalgamation of the terms phishing and pharming. Research into pharming attacks against home routers is an active area, for a detailed analysis see [SS06].
2.4.5.1 Hosts File

The hosts file is a file used on several operating systems as a performance improvement to eliminate the need to perform a DNS lookup for certain host names. It holds a list of IP address and domain name combinations for which the former is taken as the address for the latter. This can be useful for frequently used domain names as the network overhead of a DNS lookup is not required. This system was more popular in years past when internet connection speeds were slower but in modern times its use is less frequent.

Malware uses this legitimate OS feature to redirect requests to websites such as banking websites to attacker controlled addresses. Furthermore the technique is extremely easy to carry out, the malware just needs to add a line to the hosts file (or overwrite it altogether) with the malicious IP address for the website that the attacker wishes to impersonate, or in the case of security-related domains, to block.

This technique has the added benefit that the payload has a very small system imprint – merely the altered hosts file, no executable or other code is needed. However, it is still relatively easy to detect as the victim need simply look at the hosts file for rogue entries.

2.4.5.2 API Hooks

A second method to enable DNS redirection is to hook the APIs responsible for resolving domain names inside the browser or other target process. Using the MitB approach [2.1.4], the malware injects into the target process and alters the code at the API to point to its own code. The malware can then filter the domain name being looked up and choose to deny access or redirect to a different address of the attacker’s choosing.

This approach has been used by Citadel by hooking the APIs gethostbyname and getaddrinfo as described in [RS12].

2.5 Automation

As financial malware becomes more successful, sophisticated and complex there is a need to automate as much of the malware’s operation as possible. The time and resources needed to manage a botnet of tens or even hundreds of thousands of clients is considerable. Tasks like processing stolen data and attempting to use the obtained credentials to log in to victim’s bank accounts and initiate fraudulent transfers do not scale well for extremely large operations.
To solve this problem malware authors have introduced automation into many aspects of the malware's functionality and administration.

2.5.1 Automated Alerts

One of the first areas to become largely automated was the process of making the botnet administrator aware that a bot had come online. This is important information as this is when custom commands can be pushed to the infected machine. The botnet owner may command the infected machine to download and execute another file or to search the infected machine for specific documents.

When the victim logs into an account that requires a one-time password then it becomes important for the attacker to receive this information in real time. Several malware families implemented instant messaging modules that relayed this information in real time to the botnet owner, including the Jabber chat notifier plugin for Zeus described in [KS10].

2.5.2 Automatic Transfer System

Another area that has benefitted from automation is the area of fraudulent bank transfers. Logging in to a victim account and making a transfer is a slow and painstaking process that can easily be detected by the bank and the victim. The bank can detect that the user is logging in from a strange location and the victim will see the evidence of the fraudulent transfer the next time they log in. To reduce the time and effort required to make transfers and to decrease the likelihood of the transfer being detected, many malware authors have automated the bank transfer process.

This type of functionality is often called an Automatic Transfer System (ATS). It is usually implemented through the web injects [2.4.1]. The injected code often includes functionality to read the current account balance when the user logs into their bank account, and transfer a fixed percentage of the account to a dynamically retrieved mule account. The money will then be transferred on, eventually reaching the botnet owner. The injected code often has the ability to hide evidence of the transfer from the user by altering the contents of the web page.

This system does not require any interaction from the botnet owner which means it scales extremely well. [LK12] describes the use of Automatic Transfer Systems by financial malware.

2.6 Defeating Countermeasures

As financial malware has become more widespread and the losses suffered by banks have increased, countermeasures have been developed. Malware authors have responded with features
designed to defeat the bank’s countermeasures, a cycle which repeats, resulting in an arms race between defender and attacker.

2.6.1 Countermeasures

A wide variety of countermeasures have been developed by banks and other parties interested in defending against financial malware. Some are very public as they affect the way that normal users log in to their accounts and some are much less well known as their effectiveness relies on the malware authors having no knowledge of their existence. We summarise the main countermeasures in the next section.

2.6.1.1 Antivirus

Antivirus software is a basic measure used to help combat financial malware. It can detect the majority of malicious samples and block them from running on the client computer, stealing data and exfiltrating it to external servers.

Many banks have partnered with Antivirus vendors to offer free versions to the bank’s customers. Although not perfect, a machine protected with up to date Antivirus software is less likely to have active banking malware on it.

2.6.1.2 Browser Protection

In addition to Antivirus some companies have developed software specifically designed to protect the user’s browser against financial malware. An example is Rapport from Trusteer, which is offered as a free download by several banks including RBS, Natwest, Nationwide and HSBC.

2.6.1.3 Two-Factor Authentication

In order to combat simple password theft being enough to completely take over a customer’s bank account, many banks have implemented some sort of two factor authentication scheme.

There are many different implementations such as PINsentry, RSA tokens and Transaction Authorisation Numbers. Some solutions provide a hardware device that the user inserts their bank card into, some provide a hardware device that generates random numbers, some provide a mobile application that generates a random code and some solutions include SMS sending functionality.
Most of these solutions operate on the same principle: they generate some sort of one-time password that the user must combine with their own password or other code to log in to the banking website. This provides a second factor – something they know is their password and something they have is the device that generated the one-time password.

2.6.1.4 Fraud Detection

A less well-known countermeasure introduced by banks is fraud detection implemented on the back end. Banks use the large amount of data they possess on legitimate account activity to detect anomalous behaviour and block potentially suspect transactions.

There are some very simple rules that can be applied to detect fraudulent activity, such as logins from IP addresses that are located in countries that the user does not normally log in from. Other factors can be taken into consideration such as the operating system and browser that the user normally logs in from.

There are also more complicated statistical analysis based rules that banks can apply to determine that a log in or a transfer is potentially fraudulent. Some examples of fraud detection systems include FraudMap from Guardian Analytics, Fractals from Alaric and Proactive Risk Manager from ACI.

2.6.2 Workarounds

For many of the countermeasures introduced by banks and security companies, malware authors have implemented workarounds.

2.6.2.1 One-time Password Interception

Almost all of the two-factor authentication solutions that banks have implemented involve generating a one-time password that the user must enter during login. To defeat this countermeasure the malware authors must be able to intercept the one-time password and use it for their own purposes, be it logging in or initiating a fraudulent transfer.

Since the number generated by the bank’s solution is by its very nature temporal, the attacker must use it very quickly or it will expire and become useless. The most common method is to use the web injects to intercept the password and display an error message to the user while the attacker silently uses the code in the background. This principle is described in [CW05b].

There has been much creativity involving exactly what the attackers do after intercepting the one time password. Due to the time constraints there have been varying degrees of automation
involved, particularly automated alerts [2.5.1] and automated transfers out of the victim account [2.5.2].

### 2.6.2.2 Mobile Components

As mobile devices have become more advanced they are used in more and more of the places that traditional personal computers are used. This includes many aspects of online banking. Mobile devices are used as a secondary authentication factor and can also be the primary means of accessing the online bank account through a dedicated App or through the browser on the device. As a result the mobile device has also become a target for cyber criminals.

One of the first instances of financial malware incorporating mobile components was the Zeus family that included a mobile module that was designed to intercept SMS messages containing Transaction Authorisation Numbers. This was called *Zeus in the Mobile* or ZitMo [AA11]. There have also been other cases since then including *SpyEye in the Mobile* or SpitMo [MS11].

### 2.6.2.3 Anti-Anti-Virus

Since the primary mechanism for defending personal computers against malware is Antivirus software, some malware families specifically target and attempt to disable that software.

Tactics that have been attempted include killing processes belonging to defensive software, disabling services, patching executables, deleting database files and in-memory patching. A particularly novel technique was used by Vawtrak and involved using the legitimate Windows feature called *Software Restriction Policies* to prevent security software from being allowed to execute. [MM14] contains an analysis of this technique.

### 2.6.2.4 Remote Desktop

Many of the countermeasures introduced by banks revolve around detecting when a login occurs from an unusual location or computer. The malware authors have come up with a workaround to this measure that allows them to connect to the banking website through the infected machine. The malware client application acts as a reverse proxy, connecting back to the attacker’s command and control server, allowing the attacker to use the infected machine like a remote desktop server. All subsequent logins and account activity take place from the victim’s machine, thus defeating any measures the bank may take to identify suspicious logins.

The remote desktop functionality of *SpyEye* is mentioned in [AS11a].
Modularisation

Breaking up large software projects into self-contained modules that carry out their own specialised functions is a recognised practice in the software industry. As the financial malware industry has matured we have seen these kinds of principles and ideas bleed over to the malware world.

Modularisation has become a popular design approach to modern malware. There are many advantages to a modular approach over a monolithic architecture where all the functionality is one single unit. Development resources can be easily distributed with teams or individuals working on their own modules. Advanced functionality can be hidden in modules that are only deployed in certain conditions which can help hide the functionality from researchers and security companies. A modular framework also enables the possibility to open up the API to third party developers, creating a market for custom modules for the particular financial malware family.

The plugin architecture provided by SpyEye is discussed in [AS11b].
3  Extracting Actionable Data

In this section we define and give an overview of the different types of actionable data. We also explain their relevance, and what benefits extracting actionable data from financial malware can bring.

We then explore the three main areas from which actionable data can be extracted: memory, network and the disk. We explore the type of data we can extract from these areas and how it can be achieved.

3.1  Defining Actionable Data

Although the term “actionable data” is used on a day to day basis in many environments to have a very specific meaning, it is nevertheless a phrase with broad connotations. It is a term that has long been used in business to describe data related to the market that can be used to help make strategic decisions for the business. However, we are interested in the definition in the terms of the information security world.

Even within Information Security the definition of actionable data can vary according to the individuals and entities involved. For example [EU15] which is intended as a guide for exchanging and processing actionable information for national and governmental CERTs suggests:

“actionable information is used to take actions that mitigate against future threats, or help address existing compromises.”

and that:

“An implied consequence of such a definition is that different stakeholders will see different sets of information as actionable and that some of them may process information which is only actionable to others (directly or when combined with other sources of information).”

and:

“Based upon comments and observations of the expert group, information is defined as actionable when it meets five criteria: relevance, timeliness, accuracy, completeness, and ingestibility.”
Although comprehensive, this definition may not necessarily be universally applicable. For example, a particular team within an organisation may have a much narrower definition of actionable data. They may only consider a piece of information actionable if it can be directly used in their systems to help defend the organisation from cyber threats.

A different team in another organisation that is in the business of discovering and selling threat intelligence may have a broader definition and may also place a high value on obtaining the information and developing the processes that can be used to generate the actionable data that they ship as a product.

When speaking of financial malware we can extract directly actionable data but we may also need to extract other information that is needed as an intermediary step to help us extract the directly actionable data. To this end we describe the data that can be directly actionable and the intermediary data we need to find the directly actionable data.

### 3.1.1 Directly Actionable

In the context of financial malware the definition for this data ties in more closely with that above applying to CERTs. However, we can expand the scope of what data we might consider actionable as we are not only concerned with defending organisations but also in performing forensic post-mortems on compromised systems. For that reason we may consider information such as the contents of data exfiltrated over the network and the commands carried out by the malware on the infected system, actionable.

There are also other types of data that are not necessarily directly actionable but can help contribute to tracking malware campaigns and provide attribution to those behind the malware. This includes identifiers that malware authors embed into the malicious files to identify specific campaigns and re-use of encryption keys and mutex names by the same actors.

We can begin to produce a list of types of data that we may consider important to extract in our context:

- Command and control addresses
- System artefacts including file names, registry key names and value names, mutex names, password and encryption key values
- Exfiltrated data
- Executed commands
- Other network information, including HTTP request format strings
- Data modifications such as web injects, redirections, processes to log keystrokes for, etc.

### 3.1.2 Intermediary Data
We can consider intermediary data to be the information that is required in order to obtain the directly actionable data. The best example of this type of data is decryption keys as many of the directly actionable data we wish to extract will be encrypted.

We may also need to discover the offsets within memory, the file names and registry values that hold data, and the means to identify as well as decrypt network traffic belonging to the malware. For example, it is quite common for malware to store downloaded configuration data in a file on the disk or in the registry itself. The file name or the registry key name would be intermediary data as we need to discover them in order to extract their contents.

### 3.2 Relevance of Actionable Data

The relevance of any data is dependent on the needs of those consuming the data. The relevance applies to the consumer and is therefore variable. We must therefore attempt to describe the relevance of the data in as general terms as possible.

We can say data has relevance in two broad areas: to aid a forensic investigation, and to help identify and prevent further compromises.

#### 3.2.1 Aiding Investigation

Data can be used to aid the investigation of a compromise by helping to establish what actions the malware carried out on the compromised computer. This can take the form of network addresses that data may have been sent to, the data that was sent, the contents of any downloaded configuration files and any potential traces of activity that the malware may have left on the disk.

There are also other items of data that may help associate one compromise with another and therefore track malicious actors across multiple compromises. [BB13] describes the case of *Operation Crossbill* where a group using the *SpyEye* malware family were tracked using the artefacts found in malware samples and successfully prosecuted.

#### 3.2.2 Further Compromise

The second area where data has relevance is to help identify and prevent further compromises. The generally accepted approach to do this is to produce Indicators of Compromise (IoCs) that can be shared and consumed by various systems. This data consists of solid facts such as command and control addresses or hashes of malicious files, system artefacts including file names and registry keys and any other type of information that can be scanned for to determine the presence of the malware.
There are a variety of established formats for representing various aspects of this data, including STIX and Cybox developed by Mitre, OpenIOC developed by Mandiant and VERIS developed by Verizon.

### 3.3 Memory

Memory is where we must look to find items of data related to the malware while it is executing on the system. Memory has the potential to be the most fruitful resource we can use to find both directly actionable data and intermediary data.

There are a variety of methods that can be used to capture memory and a full discussion of them is outside the scope of this report. [ML14-chap4] provides a detailed analysis. However, we may capture a full dump of physical memory, or a partial dump, crash dump or a single process dump. A full physical memory dump is preferred as this will contain the most amount of information that each of the options gives us.

From memory we can obtain information such as decryption keys that the malware stores inside the address space of its process, lists of command and control addresses, and various other types of artefact that can be useful to us.

There are items of data available in memory that are simply not available from other sources, such as a network capture or may be much harder to obtain from the disk. A good example of the latter situation is decryption keys that are embedded inside the malware sample’s main executable file. This file will often be packed on disk and unpacking it can be a laborious and time consuming process. In memory, the executable will typically be unpacked so extracting information from it is a much simpler task.

#### 3.3.1 String Searching

Although primitive, string searching across memory components such as full system memory dumps, process memory dumps or crash dumps is a quick approach that can still produce useful data.

Any data that is stored in plaintext in memory may be recoverable using simple string searching. By limiting our search using careful regular expressions and other search patterns we can very effectively find certain types of data such as URLs and filenames.

However, we are limited in what data we can find using string searching. Any complex structures or encrypted, compressed or obfuscated data cannot be found using simple string searching.
3.3.2 Artefact Carving

Carving is a term used to describe the process of searching raw input for data structures and objects. When applied to memory we can use carving principles against a raw memory dump to find operating system structures including processes, files, network objects and much more.

We might use this technique to find data structures used by the malware, such as custom structures that follow a specific format and are used by the malware to store data that is relevant to us. We might also use this technique to build a picture of memory as it exists on the victim machine, for example by building up a list of processes and mapping physical memory locations to virtual memory addresses to identify the regions of memory belonging to each process. We can then isolate regions of the raw dump and act on them as if we were accessing them from within a running machine.

Software suites that achieve this type of abstraction are often referred to as memory forensics or memory analysis suites.

3.3.3 Memory Analysis Suites

Memory analysis suites typically operate against a memory dump though some can also work against live memory. They are designed to provide an easy to use interface between the complexities of physical memory and a human analyst.

3.3.3.1 Volatility

To quote [ML14-pp45]:

“

The Volatility Framework is a completely open collection of tools, implemented in Python... Analysts use Volatility for the extraction of digital artifacts from volatile memory (RAM) samples.

“

Volatility is available under the GPL v2 and as such is available free for anyone to use and modify. This has created a very active community with many community-contributed plugins as well as fast-paced development and multiple operating system and architecture support.

Volatility is a command-line tool that is well suited to automation. However, it is not necessarily a simple tool for novices and being open source does not have a paid-for support system. Its customisability and rich community resources make it an excellent tool for advanced malware analysis and ideal for extracting malware artefacts from memory.
3.3.3.2 Redline

Redline is a free tool produced by Madiant. From Mandiant’ website:

“Redline, Mandiant’s premier free tool, provides host investigative capabilities to users to find signs of malicious activity through memory and file analysis, and the development of a threat assessment profile. “

In terms of memory analysis suites Redline is used more as an Incident Response and triage tool though when used in conjunction with Mandiant’s other products can form part of a comprehensive memory analysis suite.

Although Redline has similar capabilities to Volatility it does not have the extensibility and is not open source.

3.3.3.3 HBGary Responder

Responder is a commercial incident response and memory analysis tool produced by HBGary. It is designed to be easy to use, to enable rapid workflow and to quickly produce results for incident responders.

It includes many advanced acquisition and analysis features particularly in the area of automated analysis. However, its lack of extensibility and high cost make it a less attractive choice for certain environments.

3.4 Network

Some of the most vital pieces of information, particularly when aiding an investigation, can be extracted from data that has passed over a network. This includes configuration files downloaded by the client, commands sent by the attacker and perhaps most importantly data that has been exfiltrated.

Network data can be captured using a variety of means. [KJ10-chap3] provides a thorough analysis of the different types of network data and the different methods used to capture it.

Most modern financial malware families deploy a dynamic configuration file at execution time. This file contains a wealth of valuable information including further command and control addresses, additional modules, web injections and more. Extracting this file from the network traffic of the malware can give us an excellent idea of the capabilities and intentions of the malware and those operating the malware.
Botnet administrators will often issue commands to infected machines. For financial malware these commands often centre on downloading new updates and other pieces of malware; starting a reverse shell to an attacker controlled machine; taking screenshots and other information-gathering commands. Harvesting details of any issued commands from network data can help us ascertain what operations were carried out on the infected machine.

Extracting exfiltrated data from network data has the obvious benefits that investigators can attain a full grasp of the extent of the breach and understand exactly what data was stolen.

### 3.4.1 Manual Analysis

In most cases network data will be provided by a packet capture, usually in pcap format. These captures can be manually analysed using tools such as Wireshark. Using manual tools may produce some results, including command and control addresses that the infected machine communicated with and the amount of data exchanged, and if the malware sends and receives data in the clear then we will be able to read that data with such tools. However, a manual approach will meet a dead end when encryption is applied and can become very challenging when the traffic capture contains the data from a large number of active hosts.

To help defeat the encryption or obfuscation used we can combine manual analysis with some level of automation by using manual tools to dump out sequences of the packet capture and feed them into separate tools to perform the decryption. This may be satisfactory for small packet captures but the approach is laborious and does not scale well.

### 3.4.3 Automated Protocol Analysis

When dealing with large packet captures or highly obfuscated, encrypted or compressed data we need to automate as much of the analysis process as possible. Ideally we need a system that takes as input a network capture and outputs the information we have identified as relevant.

#### 3.4.3.1 Custom Tools

A common approach to extracting useful data from network captures is to write a custom tool for the particular malware family involved. There are various programming libraries available that allow for low-level analysis of network capture data, such as libpcap.

These libraries can be used to write a tool that parses the network streams sent and received by the malware, decodes the data and extracts the relevant pieces. Although this is a viable solution for a specific case, there must always be a sizeable amount of code that must handle the
lower level protocols and when the same approach is to be applied to a different malware family much of this code must be duplicated or re-implemented.

### 3.4.3.2 Chopshop

From [WS12]:

"Chopshop is a MITRE-developed protocol decoder framework built around pynids. One of the goals of chopshop is to eliminate the mundane work that goes into writing protocol decoders, allowing decoder authors to quickly write the important parts. By abstracting away all the boilerplate code, chopshop only requires decoders to conform to a simple, easy-to-use interface."

Chopshop provides a framework where we can write code that just deals with extracting the data that is relevant to us, with the lower level details handled by the framework. Modules are written in python and can also be layered on top of each other, for example a module for decoding Zeus traffic can be layered on top of the HTTP module.

Although Chopshop is primarily used for decoding traffic related to Advanced Persistent Threats (APTs) it lends itself well to decoding traffic from financial malware and extracting data items of relevance.

### 3.5 Disk

The final significant area from which we can extract useful data is the disk. In many cases an image of the disk of the infected computer is the only resource available. The disk represents non-volatile storage for the system. Thus we will most always find the reboot persistence mechanism, which is usually an executable file, of the malware located somewhere on the disk. We may also find temporary data stores used by the malware and other meta-information about what actions the malicious program took on the infected computer.

Ideally we will deal with a bit for bit copy of the disk drive from the victim machine. This avoids making changes to the state of the machine that running an investigation against a live system would otherwise induce. [KJ10-part3] explains the various techniques that can be used to capture a full disk image.

### 3.5.1 File System
The regular file system on the machine may provide sources of actionable data. Typically the malware executable file will be stored on disk. We may search the file system or use pieces of information extracted from other sources such as memory to locate the file and retrieve it. We can then analyse this file to extract further information. We achieve this through manual reverse engineering or perhaps through an automated analysis system or sandbox.

The malware may also store modules and configuration files that it has downloaded or temporary caches of stolen data on the disk. [HN10] describes how SpyEye stores its configuration data and module files in an encrypted and obfuscated zip file on the disk.

The disk may also contain valuable traces of the activities of the malware. For example if any other commands or programs were executed there may be prefetch files left on the file system. Prefetching is a mechanism that Windows uses to help improve performance. Prefetch files contain information about the files loaded by the program. The presence of these files can help us ascertain what if any commands the attacker sent to the malware.

Depending on what state the machine was in when the disk image was taken, there may be files that can be used to build other sources of data. For example, if the computer was put into a hibernation state the hibernation file will be stored on the disk and can be used as a source of memory data.

### 3.5.2 Windows Registry

The Windows Registry is a binary database that contains settings and configuration options for the operating system itself, and also for Windows applications and any other third party applications that choose to use it (which in practice is almost all applications).

The Registry is stored on the disk as several hive files. The location of each of these files can be found from the following key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\hivelist
```

Which will usually contain data similar to:

```
REGISTRY\MACHINE \SYSTEM : \system32\config\system
REGISTRY\MACHINE \SAM : \system32\config\sam
REGISTRY\MACHINE \SECURITY : \system32\config\security
REGISTRY\MACHINE \SOFTWARE : \system32\config\software
```
Since these hive files are located on the disk we can extract the files and analyse them offline.

The Registry can contain useful meta-data about the malware such as how the malware’s executable file on disk is automatically started at system boot. This will usually be triggered by a registry entry in one of the locations used to execute applications at start up, for example:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
```

The files that are referenced by these registry locations are useful starting points when looking for potentially suspicious files and may allow us to find the malware’s main module without having any further information about the malware family.

As was mentioned in [3.5.1], malware will often download configuration files and further modules and store them in some form on the disk. Malware also stores this kind of data in the Registry, so pulling the registry hives off the disk allows us to extract and analyse this stored data.


### 3.5.3 Unallocated Blocks

Some advanced malware families store elements of their code and configuration files in sectors outside of the regular file system. In order to achieve this they typically include some element of kernel-mode rootkit [2.3.3] to access the raw disk. [AM10] describes how the TDSS rootkit creates a custom file system in unallocated blocks on the disk and stores its modules and data files inside it.

It is important when making a full disk image that any unallocated blocks are also captured, as they may contain vital information stored by the malware.
4 Demonstrating Data Extraction

In this final section we demonstrate extracting actionable data using the techniques discussed in the earlier sections for a modern and current financial malware family. We will produce a program that includes modules to extract data from each of the key sources we have identified and links the output of those modules to provide more actionable data than is possible from any of the individual data sources in isolation.

This section describes the malware family that has been selected, the data that we will attempt to extract, touches briefly on the reverse engineering process undertaken to discover how we might locate, extract and decode that data, and then we will describe each of the modules that were written to extract that data from memory, network and disk data, before finally describing the overall program used to link those modules and output the final actionable data.

4.1 Malware Selection

To be as valuable as possible the malware that we extract actionable data for must be current, prevalent and there must not be existing code that performs the same function. The project therefore provides value to the forensics, incident response and information security communities by providing re-usable code that can be employed to aid an investigation into a compromise and to help prevent future compromises.

These selection criteria rule out several families: there is existing code to extract data from memory for Zeus; SpyEye and Shylock are no longer particularly prevalent. There have been several recent industry publications describing Vawtrak which is also known as NeverQuest and Snifula, which can be found in [JW14b], [JL14], [DJ14a], [RA15], [SM13].

There has not been any code published to extract actionable data for Vawtrak, and the many publications concerning it imply it is current and relevant, making it a good candidate.

For a list of the samples used please see Appendix A.

4.1.1 Vawtrak Overview

Vawtrak is a financial malware family that follows similar principles to earlier families such as Zeus and SpyEye. It injects itself into system and browser processes, harvesting credentials and other data and is able to inject code into browsed websites on the fly. It reaches out to a command and
control server during execution, downloads a configuration file that contains the websites to be targeted and the code to inject into them, and sends stolen data back over HTTP.

Vawtrak has been employed to steal from a large number of banks throughout the world as well as non-financial targets. [DJ14b] describes how Vawtrak was used to defraud StubHub which is an eBay owned events-ticket website to the tune of 1.6 million dollars.

4.1.2 Extraction Targets

Before attempting to extract any kind of data we must first establish exactly what data it is we are looking to extract. There are several pieces of data that we would like to extract from any financial malware family of this nature.

1. **Command and control addresses.** These are an essential item to extract from any financial malware family. We also know from the works referenced in [4.1] that Vawtrak has multiple back up command and control addresses so we would like to extract all of them.

2. **Downloaded configuration file.** Financial malware often downloads a configuration file during execution that contains a wide variety of useful information.

3. **Exfiltrated data.** We would like to extract the details of any stolen data that was successfully exfiltrated.

4. **Commands sent to client.** We would like to know the details of any commands that the attackers issued to the infected client.

There are several other items of data specific to Vawtrak that we wish to extract:

5. **POST request format string.** A format string is embedded in the Vawtrak executables that specifies how it constructs the URL used to communicate with its command and control server. Extracting this string helps us identify Vawtrak network traffic even if the command and control address is not known.

6. **Project ID.** This is a numeric value used to distinguish the specific campaign that the client program is part of. This information can be used to help track infections across the campaign and may help provide additional information on the attacker’s intentions if we are missing other elements of the source data such as a network capture.

7. **Update Version.** A numeric ID used to distinguish between build versions of the client program.

4.2 Reverse Engineering

To find out how we can extract the data we have identified as required we must reverse engineer a sample of the malware.
4.2.1 Dynamic Analysis

The best place to begin when attempting to ascertain how a sample works is often to execute it and observe its behaviour. This is referred to as a component of *dynamic analysis*. We can execute a sample of the malware, capture any network activity observed, capture volatile memory and image the disk. This then provides the three raw data materials that we will use to extract data from.

Starting with the network capture we can see a series of HTTP POST requests and responses, as shown in (figure 5).

![POST /collection/80090042/02/C7704535 HTTP/1.1](image)

Both the data sent by the client and the data received from the server appear to be encrypted, although at the end of the first response from the server there is a URL that is not
encrypted. This data does not seem trivial to decipher so we will have to establish how the data is encoded from further analysis of the sample.

We can also observe the registry and disk changes made during the execution of the sample. We observe the following suspicious file is dropped:

"C:\ProgramData\CokjaTpopa\XusvImluy.qxg"

The following registry entry is created that will cause the suspicious file to be executed at system start up:

Key: HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run

ValueName: CokjaTpopa

Value: regsvr32.exe "C:\ProgramData\CokjaTpopa\XusvImluy.qxg"

We also observe the following registry entry created with a large amount of binary data:

HKEY_CURRENT_USER\SOFTWARE\{554612FB-7CA4-45BC-8CE5-720E6CDF2EF3}\n
We can make some reasoned guesses about these data items based on what we already know of Vawtrak:

- The dropped file is most likely the main Vawtrak module and the runkey entry will ensure it is executed after each system boot.
- The registry entry with binary data stored in it may well be the downloaded configuration file.

Although dynamic analysis has given us some information it has not provided us with any real answers. So far the only substantial piece of information we have is the command and control address that we saw HTTP traffic going to.

To obtain further information we must look at the Vawtrak binary itself.
4.2.2 Unpacking

The executable files for the majority of modern malware are obfuscated in some way in a process known as packing. The process encrypts, compresses, or otherwise renders the genuine contents of the file unreadable. This process is designed to make it much harder to understand what the sample does and to attempt to evade detection from antivirus software. The code belonging to the packer decrypts the main body of the malware and executes it.

Reversing this process, or unpacking, can be a very slow and laborious task. [RA15] describes the process of unpacking a Vawtrak sample that was packed with one particular runtime packer.

[figure 6] shows a snippet of the output from running the strings command against a packed Vawtrak sample.

```
0x00CD9C: tD^#eL
0x00D1AC: DeHgV}{e
0x00DB1C: %F`dojj
0x00DB9F: g%PXcDtyik
0x00DF32: P*$ngTn
0x00E8B: qa3iQ/
0x00E94E: Sx]{Tw
0x00E409: J&O2Ad
0x00E5C0: pQl{m=
0x00EC2E: 9mzPH9
0x00ECB8: __uyuU!
0x00F755: p$ufaG-qC
0x00FEB8: h~aH8k
0x00FFC2: Ud6fqc
0x010237: b>W,Nl3
0x0104C1: "Gyqe)
```

[figure 6 - packed sample strings]

Fortunately, banking malware like Vawtrak often injects itself into other processes on the system, and the code that is injected is almost always unpacked code. We can use this fact to examine the unpacked code by dumping from memory.

After capturing physical memory from the infected machine we can use a Volatility plugin called malfind to extract the injected code. Malfind looks for memory regions that have the characteristics of injected code and dumps them out. It is described by the author in [ML09].

Running strings against the dumped file gives us much more encouraging results, as shown in [figure 7].
4.2.3 Finding Relevant Code
Now that we have an unpacked sample to analyse we must locate the pieces of code that deal with the data that we are interested in. For this we need to load the file in a disassembler. The de-facto industry choice is the IDA Pro disassembler produced by Hex-Rays.

IDA presents us with a disassembled view of the program’s machine code, as seen in [figure 8].

![Disassembled view](image)

This is not a decompiled representation, but a representation of the assembly code that higher level languages such as C and C++ are compiled into. Establishing exactly what the code does is a much more challenging task than simply reading program source code.

Our first aim is to find the command and control addresses and establish the encryption used to encode the POST data and to decipher the responses. We can begin by seeing if any web addresses appear as strings in the unpacked binary, particularly the address we saw in the network capture. Unfortunately they do not exist in plain text so we must search for other strings that may lead us to the area of code where the addresses are decoded.
Fortunately there are a wide variety of different strings inside the file that look to be related to the network communications, [figure 9] shows a subset.

[figure 9 - network strings]

Finding these strings in IDA leads us to the portion of code that uses them in [figure 10].

[figure 10 - post reference]

Tracing backwards through the code leads us to the main decryption function and the loop where the command and control addresses are stored.

After further analysis we find a structure that contains the data we are interested in. It contains the Project ID, the Update Version, and the format string for the POST request and the command and control address in encrypted form. We can use the decryption routine we identified to decrypt the encrypted data. For details of the full algorithm see Appendix B.
We can apply similar principles to find the algorithms to decode the various other pieces of information we are interested in. A detailed description of the complete reverse engineering of Vawtrak is beyond the scope of this report.

4.3 Volatility Module

Now that we know where Vawtrak stores the information we require in memory and how to decrypt it, we must implement that logic in an effective and re-usable way. The mechanism chosen to achieve that is through a Volatility module.

Volatility was described in [3.3.3.1] and is a framework for analysing memory. Volatility is open-source and written in Python. We will write a module that finds the Vawtrak code in memory, and the data structure where the encrypted data is stored, decrypts it and presents the decrypted data to the user.

4.3.1 Previous Work

Volatility is an ideal framework to build this kind of logic. Indeed, although no module exists to extract data for Vawtrak, modules do exist to extract data for other malware families including Zeus called zeusscan described in [ML11]. Our module will follow a similar framework to zeusscan, and all other Volatility modules but since they are two completely different malware families all the implementation code will be very different.

4.3.2 Vawtrakscan

The Volatility module consists of the following key stages:

- Filter. This stage filters the memory regions so that we are left with only the regions likely to hold the injected Vawtrak code.
- Matching. This stage runs a set of Yara signatures against the region to find the addresses of the code blocks that reference the command and control address structure.
- Decryption. This stage decrypts the data structures and stores and structures the decrypted data.
- Render. This stage takes the gathered data and presents it to the user.

[figure 11] shows a code snippet from vawtrakscan.
def calculate(self):
    if not has_yara:
        debug.error("You must install yara")

    addr_space = utils.load_as(self.config)

    if not self.is_valid_profile(addr_space.profile):
        debug.error("This command does not support the selected profile.")

    rules = yara.compile(sources = self.signatures)

    for task in self.filter_tasks(tasks.pslist(addr_space)):
        task_space = task.get_process_address_space()

        # We must have a process AS
        if not task_space:
            continue

        for vad, process_space in task.get_vads():

            if obj.Object("_IMAGE_DOS_HEADER", offset = vad.Start,
                           vm = process_space).e_magic != 0x5A4D:
                continue

            debug.debug("Got DOS header")

            if vad.Length > 0x500000:
                continue

[figure 11 - vawtrakscan snippet]

The result is a text output of the extracted data, as shown in [figure 12].

[figure 12 - vawtrakscan output]
Since Volatility is written in Python we can import it as a module into other applications. This means we can write a program that uses Volatility and have the output of the vawtrakscan module available as a data structure for further processing.

4.4 Chopshop Module

To decode the network traffic we have implemented a Chopshop module. As we described in [3.4.3.2] Chopshop is a protocol analysis framework that allows us to concentrate on writing code that actually deals with the payload data, with the framework itself handling the complications of the underlying protocols.

Our Chopshop module will decipher data sent by the client program and received from the server.

4.4.1 Previous Work

The source code repository for Chopshop on Github also includes several modules to dissect the network protocols of several malware families. These include EtumBot, Gh0st, NetWire and Poisonivy. However, they do not include any financial malware such as Vawtrak. Nevertheless, existing modules are useful as a guide on how to go about writing a Chopshop module.

4.4.2 Vawtrak Module

Vawtrak communicates with its command and control server over HTTP so our module must operate on HTTP data. Chopshop includes a facility called “module chaining” which allows the output of one module to be piped into another module. So for our purposes we can chain the output of the HTTP module into our Vawtrak module. This is accomplished using the pipe operator – ‘|’. We also give our module a list of host names that we wish to examine the traffic for. Here is an example:

```
./chopshop –f <pcap file> “http | vawtrak –H host1.com,host2.com”
```

The Vawtrak Chopshop module has the following significant sections:

- Initialisation. Here we process the host name arguments and create the “cmd_codes” table which maps the numeric value received from the command and control server that indicates a command to be executed by the client, with the routine that processes that command. A snippet can be seen in [figure 13].
handleProtocol. This is the main control flow routine where the incoming data is processed. We examine if this is data sent by the client or received from the server, establish what type of data this is, such as configuration file data or received commands, and decrypt and process the data with the appropriate routine.

Chopshop allows for multiple different output options including text and JSON.
4.5 The Sleuth Kit

The Sleuth Kit (TSK) is a collection of tools written by Brian Carrier for analysing disk and file system images for forensic evidence. It is described in [BC07-p.537-543]. TSK is commonly used in forensic investigations as it has very powerful capabilities.

Our needs are relatively simple as we are simply looking to extract specific files from a disk image. We use TSK to achieve this as it is open-source and abundantly capable of fulfilling our needs.

4.5.1 Extracting Files with TSK

We can extract specific files from a disk image using the *ifind* and *icat* TSK commands. *icat* is used to dump a file from the image given its inode address. We use *ifind* to get the inode address of a file given its filename.

4.6 Putting it all Together

As we have demonstrated in the previous sections we can extract useful data from different sources in isolation but we achieve a much more complete picture by combining the data from memory, the network and the disk together.

To this end we designed a program in Python called *ExtractData* that takes as input a configuration file that details the locations of the memory image, network capture file, and disk image to work on, and presents all of the extracted data in a digestible form that is suitable for consumption by other systems.

4.6.1 Design

*ExtractData* is primarily focused on running the various modules that do the actual data extraction, collating the data and feeding it into later modules, and then presenting that final data at the end. It must be launched from Linux as some of the tools that it depends on (Chopshop, TSK) require Linux.

*ExtractData* is broken down into the following subsections:

- Get the configuration.
- Process memory.
- Process network.
- Process disk.
The configuration file is broken down into sections that detail the name of the malware family being analysed, and the locations of the files for each data source and any other options for each item. [Figure 14] shows an example configuration file.

```plaintext
# sample config file for use with extract.py

4 [family]
5 # threat family name that we wish to extract the data for
6 name = vawtrak

8 [memory]
9 # file name of the memory dump, can be full path or relative to the config file
10 dump_file = memory.dmp
11 osprofile = Win7SP1x86

13 [network]
14 # file name of the pcap file
15 pcap_file = dump.pcap

17 [disk]
18 # file name of disk image
19 disk_image = disk.dd
```

[Figure 14 - sample config]

After gathering the configuration options the memory data is extracted. Memory is processed first as we will normally need data from memory to process the other areas, though a future improvement would be to make the order of processing customisable. The memory processing routine takes the family name and the memory options from the configuration file and runs the appropriate Volatility plugin (in our case the `vawtrakscan` plugin discussed in [4.3.2]) against the memory image. The extracted data is preserved for use in the remaining sections.

The next stage is the network processing stage. The `vawtrak` Chopshop module is called with the data extracted from the memory processing stage fed into it. All the extracted network traffic is output into a JSON file for further processing from the later stages.

The next stage is to extract data from the disk. Unfortunately, due to time constraints, the implementation of this section was not completed. However, the design allowed for searching and extraction of file system artefacts based on file name.

The final stage is to present the extracted data to the user. This is achieved through text output but the data is also output in JSON so it can be used by other systems.
4.6.2 Using

ExtractData has relatively simple usage. Once the config.ini has been produced we pass its location to extract.py and give it a directory to output extracted data into. [figure15] shows the usage message.

![ExtractData usage](image)

To successfully extract data from memory Volatility must already be installed and for network data Chopshop must already be installed. Their paths must be included in the %PYTHONPATH environment variable.

4.6.3 Output

The output provided by extract.py is a summary of all the extracted data. For the Vawtrak implementation we output the extraction targets from memory: the POST request format string, the update version, the project ID and the list of command and control servers; from the network data we output details of received commands and which URL delivered them, and details of any downloaded configuration files. We also output this information into a JSON file in the output directory. [figure 16] shows the extracted data from memory, [figure 17] shows the extracted received commands from the network, [figure 18] shows a decoded configuration file extracted from the network traffic.
Displaying extracted data for threat family: vawtrak

Memory Artifacts:
format_strings: ['/collection/{PROJECT_ID:HD}/TYPE:HB}/BOT_ID:HD}]
update_vers: [12]
project_ids: [56]
c2_list: ['segropa.com', 'zaslova.com', 'geribed.com', 'dotrabu.com',

[figure 16 - extracted data from memory]

CMD received:

CMD_NAME: UPDATE_NO_REBOOT
HOST: segropa.com
URI: /collection/00000042/00/C7704535
DATA_TYPE: CMD
ARG: http://dedorka.com/upd/66?id=3346023733&n=12&n=15

CMD received:

CMD_NAME: PROCESS_LIST
HOST: segropa.com
URI: /collection/00000042/00/C7704535
DATA_TYPE: CMD
ARG:

CMD received:

CMD_NAME: GRAB_PONY
HOST: segropa.com
URI: /collection/00000042/00/C7704535
DATA_TYPE: CMD
ARG:

finished extracting data

[figure 17 - extracted cmds received]
Although time did not allow for the implementation to be completed, at this stage we would also output details of exfiltrated data.

4.6.4 Effectiveness

To measure the effectiveness of our solution let us recap the data extraction targets and how well we achieved against them:

1. **Command and control addresses.** Complete
2. **Downloaded configuration file.** Complete
3. **Exfiltrated data.** Incomplete
4. **Commands sent to client.** Complete
5. **POST request format string.** Complete
6. **Project ID.** Complete
7. **Update Version.** Complete

This represents an 85% success rate against our data extraction targets. We can also measure the effectiveness against how many different samples our solution succeeds for. Although it is very difficult to provide conclusive data as the number of Vawtrak samples is unknown, data extraction succeeds for all of the sample set attempted from Appendix A.

We can also attempt to measure how easily data extraction for financial malware families other than Vawtrak can be added to the solution. This is simply a case of implementing a Volatility module and a Chopshop module for the new family. *ExtractData* can easily be extended to handle new
malware families, however, implementing the respective modules for Volatility and Chopshop represent the bulk of the work required and will not always be easy tasks.

5 Conclusion

The aims of this project were to understand what financial malware tries to achieve and how it goes about doing that. We also aimed to establish what data we can extract from financial malware that will allow us to more deeply understand the intentions of those behind it, so that we may combat the threat they pose. Finally, we aimed to provide a practical demonstration of extracting that data.

Although the primary goal of financial malware is to facilitate online banking fraud there are a wide variety of steps it must take in order to achieve that. There are also a large number of techniques that can be used to achieve the same goals. Malware has evolved over the last decade, laying aside techniques that no longer work and developing new techniques as banks, security vendors, operating systems and browsers have all recognised the threat from financial malware and attempted to combat it.

However, there are consistent themes across the last decade of financial malware. In some form or another there has always been a desire to deliver the data that is required to commit fraud into the hands of those operating the malware. That data must be elicited from the victim, captured, and delivered to the attackers.

Actionable data can take a variety of forms. Some data is more useful for identifying existing infections, such as command and control addresses and other indicators of compromise, other types of data is more useful during a forensic investigation to assess the impact of a breach.

Although elements of this data may be extracted relatively simply, the majority requires considerable time and skill. For this reason it is important to distil the knowledge required to extract that data into a form that can be re-used in a simple and extendable way.

The practical elements of this report show that re-usable frameworks are a viable tool to extract actionable data for financial malware. By abstracting away the complexities of dealing with the source medium of the data they leave the analyst free to concentrate on the malware itself.

5.1 Future Work

The first obvious area of future work is to complete the data extraction picture for the Vawtrak family. There are more elements of the network communications that can be decoded, such as data exfiltration, and there may be more elements of data that can be found in memory such as registry data that is cached in memory.
The disk data extraction component can be completed so that we can pull items of interest off a captured disk image.

More malware families can be added by writing a Volatility and Chopshop module for them.

The framework can be extended by incorporating it into a larger workflow system, such as a forensic analysis system or even by adding it into an automated malware analysis platform.
Bibliography


[HS05]  Hiroshi Shinotsuka: Advances in trojan threats, PacSec 2005, slide 17 (http://pacsec.jp/psj05/psj05-shinotsuka-en.ppt)


[TP00] Tor: Hidden Service Protocol : The Tor Project, https://www.torproject.org/docs(hidden-services.html.en


Appendices

Appendix A

Vawtrak sample list, sha1 hashes:
8b2c5b3688a6288b5315f3d0c15aa26123875321
1ce64b31ed806d2085bb89cb2af9a8698e53fe58
30f6395ae14967088b4873162c368aaff07a28b
1295a3a63278f82bdc052bc3b48967c4cb518263

Appendix B

Decryption Algorithms:

Command and control address structure:

Command and control addresses are stored in 0x40 byte blocks, preceded by a 4 byte seed value.

Then perform the following on each block:

for enc_byte in enc_data:
    next = (seed * 0x343fd) & 0xffffffff
    next = (next + 0x269ec3) & 0xffffffff
    seed = next
    key_byte = (next >> 0x10) & 0xffffffff
    key_byte = key_byte & 0x7fff
    out[index] = ord(enc_byte) ^ (key_byte & 0xff)
    index += 1

Downloaded Configuration File:

Downloaded configuration data is decrypted using the same algorithm as the command and control address block, but it must then be decompressed using aPLib.