SEVENTH FRAMEWORK PROGRAMME

Information & Communication Technologies
Trustworthy ICT

NETWORK OF EXCELLENCE

A European Network of Excellence in Managing Threats and Vulnerabilities in the Future Internet: *Europe for the World* [†]

Deliverable D5.4: Intermediate Report on Internet Fraud

Abstract: This deliverable presents an overview of the current state of Internet fraud based on the research and analysis conducted in the context of WP5. It also presents the current research being performed by the *SysSec* consortium to gather and analyze data on the Internet fraud phenomena.

Contractual Date of Delivery	August 2013
Actual Date of Delivery	September 2013
Deliverable Dissemination Level	Public
Editor	Stefano Zanero, Christian Platzer
Contributors	All *SysSec* partners
Quality Assurance	Davide Balzarotti, Herbert Bos

The *SysSec* consortium consists of:

FORTH-ICS	Coordinator	Greece
Politecnico Di Milano	Principal Contractor	Italy
Vrije Universiteit Amsterdam	Principal Contractor	The Netherlands
Institut Eurécom	Principal Contractor	France
IICT-BAS	Principal Contractor	Bulgaria
Technical University of Vienna	Principal Contractor	Austria
Chalmers University	Principal Contractor	Sweden
TUBITAK-BILGEM	Principal Contractor	Turkey

[†] The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 257007.

Contents

List of Figures

1 Introduction

In the current digital economy, cyber-crime is ubiquitous and has become a major security issue. New attacks and business models appear every year [74, 51] and criminals keep improving their techniques to trap their victims in order to achieve their, usually financial, goals.

Modern cyber criminals are widely recognized to be well-organized and profit-driven, as opposed to the reputation-driven underground which was prevalent years ago [46].

In this Deliverable, we wish to summarize and report recent research directions that compose the state of the art in the analysis of Internet-related fraud activities.

We will start by analyzing, in Chapter 2, the complex phenomenon of Internet banking and credit card fraud, the current approaches to perform anomaly detection and fraud analysis, and the state of the art and weak points of current research in this important area.

In Chapter 3 we will drill down on the specific sub-threat of information stealing trojans, and introduce two approaches we developed to analyze them.

As a part of their arsenal, the miscreants have learned to streamline their campaigns also by leveraging automated social engineering attacks over several channels including emails, instant messaging, social networks [63], and the phone system (with both calls and text messages), with the common goal of expanding their "business" beyond email users. While the role of other features in illegal online activities has been extensively studied [83, 119, 78, 50, 45], the role of phone numbers has been largely ignored. In Chapter 4 we analyze this often overlooked component of modern Internet-based fraud, and report on two sets of experiments we are currently conducting.

Finally, in Chapter 5 we address the growing importance of data found in online social network profiles for solving criminal investigations related

to Internet fraud, and introduce a modular investigation framework that targets popular online social networks.

2

Overview of current research on online banking and payment card fraud

In this chapter we cover the current research on online banking fraud and anomaly detection. First we give a description of the anomaly detection theory necessary to understand the analysis, describing the terminologies used and the different faces of the anomaly detection problem.

For a more complete reference, we refer the reader to [27], Chapter 10.

2.1 Main Threats to Internet Banking

To develop a detection system and, in general, a security system it is necessary to understand what kind of dangers threaten the field under analysis. In the light of this fact in this section we give an overview of the major threats related to Internet banking.

Phishing With the term phishing we refer to the act of attempting to acquire information such as usernames, passwords, and credit card details (and sometimes, indirectly, money) by masquerading as a trustworthy entity in an electronic communication ([101], [120]). Although this information can also be obtained through less sophisticated means (e.g., eavesdropping, guessing, dumpster diving, and shoulder-surfing), phishing is a common form of cybercrime typically carried out through e-mail, instant messaging, and social networking sites. Phishing attacks usually provides links or instructions that direct the user to a fake website whose look and feel are almost identical to the legitimate one. The unsuspecting users are infected with malware and enter personal information in forms (such as user names, passwords, Social Security Numbers, and credit card/account numbers), which are then collected by the criminal.

Password Stealing and Identity Theft With the terms "identity theft" and "password stealing" we refer to all types of crime in which someone illicitly obtains and uses another person's personal data or password through deception or fraud, typically for monetary gain. With enough personal information about an individual, a criminal can assume that individual's identity to carry out a wide range of crimes. Identity theft occurs through several methods from very low-tech means, such as check forgery and mail theft to more high-tech schemes, such as computer spyware and social network data mining. These types of attacks rely on the ability of the attacker to fool users into giving up their personal information and credentials. Since users are typically vulnerable to these types of attacks, any method that relies on a credential that can be disclosed is vulnerable to social engineering attacks.

Password Database Theft With the term "password database theft" we refer to the attacks that steal user data and passwords from one web site operator to hack other sites. In fact stolen user credentials are a valuable commodity and, often, this information are obtained and sold. Since many people use the same user ID and password combination for multiple sites, the attacker can hack additional accounts owned by the user.

MITM With the term MITM we refer to a form of active eavesdropping in which the attacker makes independent connections with the endpoints of the connection and relays messages between them. Even if the endpoints believe that they are talking directly to each other, the entire conversation is controlled by the attacker. This attacks the mutual authentication and is only successful when the fraudster can impersonate each endpoint without the other noticing. In this type of threat, the attacker intercept all messages going between the two victims and may actively inject messages of its own into the traffic between the user's machine and the authenticating server. One approach for MITM attacks involves the usage of malicious network infrastructures (e.g., malicious wireless access points, compromised DNS servers) to redirect users from the legitimate site they are trying to access to a malicious fraudulent Web site that accesses the user credentials and acts on behalf of the user to perform malicious activities. Most cryptographic protocols include some form of endpoint authentication specifically to prevent MITM attacks. For example, SSL can authenticate one or both parties using a mutually trusted certification authority. The use of SSL authentication using a mutually trusted certification authority provides strong protection against MITM threats. Although SSL with server authentication makes man-in-the-middle attacks harder to carry out, they can fail. In example, when the certificate validation relies on

the user, the user may fail to reject fake server certificates and will click through the warning messages. There exists other methods such as OTP or smart cards/tokens for secure two-factor authentication, which limit the damage but the vulnerability still exists.

2.2 The Anomaly and Fraud Detection Problem

Many anomaly detection techniques have been specifically developed for specific application fields, while others are more generic. Anomaly detection has been the topic of a number of surveys and review articles, as well as books. From some of these (i.e., [96, 92, 37, 43]) we have extracted this overview.

Fraud can be defined as the "abuse of a profit organization's system without that abuse leading necessarily to legal consequences" as in [96], or "the use of false representations to gain an unjust advantage" ([37]). We can generalize the definition to fraud being the usage of a system in a way which does not conform to its intended rules and purposes, but close enough to legal use that the chance to go undetected and therefore not prosecuted is high.

The relationship of this issue with anomaly detection techniques is very evident. With the term anomaly detection we refer to the problem of finding patterns in data that do not conform to the expected behavior ([43]). Its importance is a consequence of the fact that anomalies in data often are not only unimportant noise, but often refer to significant information in a wide variety of application domains. In particular, anomalies may be induced in the data by malicious activities (e.g., credit card fraud, cyber-intrusion, or breakdown of a system).

A straightforward anomaly detection approach is to define a region representing normal behavior, and declare any observation that does not belong to this normal region as an anomaly. But there are several factors that make the anomaly detection problem increasingly challenging.

First, it is very difficult to define a normal region that encompasses every possible normal behavior. In fact the boundary between normal and anomalous behavior is often imprecise. For example, when anomalies are the result of malicious actions, the fraudsters often adapt themselves to make the anomalous observations appear normal, thereby making the task of defining normal behavior more and more difficult.

Second, in many domains (e.g., intrusion detection) normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future. In addition, those changes might be mistakenly identified as outliers. This highlights the need of a model able to deal with concept drift.

Another important challenge is represented by the fact that an exact notion of anomaly is different for different application domains. Thus applying a technique developed in one domain to another is not straightforward.

The final challenge resides in the quality of data. In fact, often real data contains noise similar to the actual anomalies and hence difficult to distinguish. In addition, the scarcity of labeled data for training and validation of models is usually a major issue and imposes limitations on the results and conclusions reached.

Due to these challenges, the anomaly detection problem, in its most general form, is not easy to solve and researchers have adopted concepts from diverse disciplines such as statistics, machine learning, data mining, and information theory to apply them to specific problem formulations. The formulation is induced by various factors such as the nature of the data, availability of labeled data, type of anomalies to be detected, and more often by the application domain. In other words the anomaly detection techniques needs to be adapted to different application domains.

2.2.1 Types of Anomaly

Anomalies can be classified into either point anomalies, contextual anomalies or collective anomalies.

Point Anomalies An individual data instance is termed as a point anomaly when it is anomalous with respect to the rest of data. This is the simplest type of anomaly and is the focus of the majority of research on anomaly detection (e.g., Credit Card Fraud detection).

Contextual Anomalies A data instance is termed as a contextual anomaly when it is anomalous in a specific context, but not otherwise. The notion of a context is induced by the structure in the data set and has to be specified as a part of the problem formulation. Each data instance is defined using contextual and behavioral attributes ([43]). The contextual attributes are used to determine the context (or neighbourhood) for that instance, while behavioral attributes define the non contextual characteristics of an instance. For example, in a spatial data sets the longitude and latitude of a location might be a contextual attributes while the amount of rainfall at any location might be a behavioral attribute. The anomalous behavior is determined using the values of the behavioral attributes within a specific context. A data instance might be a contextual anomaly in a given context, but an identical data instance (in terms of behavioral attributes) could be considered normal in a different context. Identifying contextual and behavioral attributes is a key requirement for a contextual anomaly detection technique.

Collective Anomalies A collection of related data instances is termed as a collective anomaly when is anomalous with respect to the entire data set. The individual data instances in a collective anomaly may not be anomalies by themselves, but their occurrence together is anomalous. Clearly this can only occur in data sets where the records are related to each other.

2.2.2 Approaches to Anomaly Detection

There are many approaches to anomaly detection that depends on various aspects:

- The type of anomaly
- The type of anomaly detection and challenges associated
- The type of dataset and attributes
- The type of existing techniques

From the point of view of the type of the anomaly the approaches could either be global or local. Global approaches refer to the techniques in which the anomaly score is assigned to each instance with respect to the entire data set. Instead, local approaches assign the anomaly score with respect to its neighbourhood. The local approaches can detect outliers that can not be detected with the global methods. In [36] a slightly different approach is implemented. The implemented tool detects individual objects that begin to behave in a way distinct from objects to which they had previously been similar. Each object is selected as a target object and is compared with all other objects in the database, using comparison criteria that summarise behavior patterns of each object. Based on this comparison, a peer group of objects most similar to the target object is chosen. The behavior of the peer group is then summarised at each subsequent time point, and the behavior of the target object is compared with the summary of its peer group. Those target objects exhibiting behavior most different from their peer group summary behavior are flagged as meriting closer investigation.

Another classification of anomaly detection techniques can be made on the basis of when the anomaly detection is performed. The detection can be online or offline. Online mode detect anomalies as soon as new data are introduced in input. That is, every time a datum is input, the system evaluate how much the datum deviated with respect to a normal pattern. In contrast, most existing works on outlier detection, are concerned with batch-detection or offline processes, in which outliers are detected only after the analysis of the entire dataset. The online setting is more realistic in such situations as Online Banking where data becomes available over time and it is important to identify deviations as soon as they arise.

Anomaly detection techniques can be divided according to the type of the training dataset and, in particular, to the fact that its instances are labeled or not. Labels associated with a data instance denote whether that instance is normal or anomalous. It is important to note that obtaining labeled data that is accurate and representative of all types of behaviors, is often very expensive, because this has to be done manually by a human expert. Typically, getting a labeled set of anomalous data instances that covers all possible type of anomalous behavior is more difficult than getting labels for normal behavior. Moreover, the anomalous behavior is often dynamic in nature. For example, new types of anomalies might arise, for which there is no labeled training data.

In the light of these facts, anomaly detection techniques can operate in one of the following three modalities:

- **Supervised Anomaly Detection**: techniques trained in supervised mode assume the availability of a training data set that has labeled instances for normal as well as anomaly classes. A typical approach is divided into two main phases: training and testing. The training phase build a predictive model for normal versus anomaly classes. In the testing phase any unseen data instance is compared against the trained model to determine which class it belongs to.

- **Semi-supervised Anomaly Detection**: techniques that operate in a semi-supervised mode, assume that the training data has labeled instances only for the normal class. Since they do not require labels for the anomaly class, they are more widely applicable than supervised techniques. The typical approach used in such techniques is to build a model for the class corresponding to normal behavior, and use the model to identify anomalies in the test data.

- **Unsupervised Anomaly Detection**: techniques that operate in unsupervised mode do not require training data, and thus are most widely applicable. The techniques in this category make the implicit assumption that normal instances are far more frequent than anomalies. If this assumption is not true, then such techniques suffer from high false alarm rate. Unsupervised techniques can be adapted for online detection upon the availability of a sufficiently large database which would enable comparison with new data.

Many semi-supervised techniques can be adapted to operate in an unsupervised mode by using a sample of the unlabeled data set as training data (e.g., [43, 93]). Such adaptation assumes that the test data contains very few anomalies and the model learned during training is robust to these few anomalies. Although the supervised learning based approach is popular in fraud detection, the semi-supervised and unsupervised learning approach

are not only more technically difficult but also more practically important, since in real situations fraud labeled examples might not be available, and a new fraud or intrusion pattern which didn't appear in past data may possibly emerge in future data.

An important aspect for any anomaly detection technique is the manner in which the anomalies are reported. Typically, the outputs produced by anomaly detection techniques are one of the following two types ([43]):

- **Scores**: scoring techniques assign an anomaly score to each instance in the test data depending on the degree to which that instance is considered an anomaly. Thus the output of such techniques is a ranked list of anomalies. An analyst may choose to either analyse the top few anomalies or use a cut-off threshold to select the most anomalous instance.

- **Labels**: techniques in this category assign a label (normal or anomalous) to each test instance.

Scoring-based anomaly detection techniques allow the analyst to use a domain specific threshold to select the most relevant anomalies. Techniques that provide binary labels to the test instances do not directly allow the analysts to make such a choice, though this can be controlled indirectly by tuning parameters specific of each technique.

On the basis of the existing techniques, anomaly detection methods can be grouped into one of the following main categories ([43]):

- **Classification based algorithms** are mainly supervised algorithms that assumes that the distinction between anomalous and normal instances can be modelled for a particular feature space.

- **Nearest-neighbour based algorithms** assume that anomalies lie in sparse neighbourhoods and that they are distant from their nearest neighbours. They are mainly unsupervised algorithms.

- **Clustering based algorithms** work by grouping similar objects into clusters and assume that anomalies either do not belong to any cluster, or that they are distant from their cluster centres or that they belong to small and sparse clusters. The majority are unsupervised algorithms.

- **Statistical approaches** label objects as anomalies if they deviate from the assumed stochastic model.

2.3 Online Banking Fraud and Detection Systems Characteristics

In this section we focus on the specific characteristics of online banking fraud. Fraudulent online banking activities are becoming more and more sophisticated, seriously threatening the security and trust of online banking business. Online banking fraud has become a serious issue in financial crime management for all banks. It is becoming ever more challenging and leads to massive losses, due to the emergence and evolution of sophisticated online banking fraud. In fact the more a business switches to online transfers, the more it is exposed to potential frauds. Frauds is becoming a relevant or even critical problem, because even if only a small percentage of the transactions committed each day is fraudulent, this can have a big impact not only on the financial side, but also on the reputational side. A bank not capable of detecting fraudulent behavior in its transactions, can be accused of not being careful enough and a loss of customer trust can have an impact much bigger than only the financial loss caused by the fraudulent transactions.

2.3.1 Online Banking Fraud

In our research, we found out that real-world online banking transaction datasets and the detection of online banking frauds have relevant characteristics and challenges which are to be necessarily considered for designing detection systems.

Online banking data involve a large number of transactions, usually millions per years, but the number of daily frauds is usually very small ([122]). For this reason the detection of rare fraud dispersed among a massive number of genuine transactions requires a high level of efficiency and needs to be real time. In other words a fraud detection alert should be generated as quickly as possible to prevent money loss.

A second feature is that the fraud behavior is dynamic. Fraudsters continually advance and change their techniques to contrast online banking defences. Malware, which accounts for the greater part of online banking fraud, has been reported in 2013 to have over 200,000 new malicious programs every day ([31]). This puts fraud detection in the position of having to defend against an ever-growing set of attacks and requires models to be adaptive. In addition it opens the possibility of using multiple models ([42]) for challenges that cannot be handled by a single model.

Another feature is that the “forensic” evidence for fraud detection is weak. For online banking transactions, it is only possible to know source accounts, destination accounts and amount associated with each transaction, but other information, for example, the profile associated to the user, its spending pattern, and its general behavior with respect to the online ser-

vice are absent. In fact in fraud detection, only the online banking activities recorded in banking systems can be accessed while information related to the compromise process are absent or difficult to obtain. This makes the process of identifying sophisticated fraud very challenging.

A further challenge is the fact that the customer habits and behavioral patterns are diverse. In conducting online banking business, every customer may perform very different transactions for different purposes. This leads to a diversity of genuine customer transactions. Furthermore fraudsters simulate genuine customer behaviors and change their behavior frequently to compete with advances in fraud detection. This means that it is very difficult to characterise frauds and even more difficult to distinguish them from genuine behaviors.

2.3.2 Online Banking Detection Systems

Fraud detection consists in identifying such unauthorised activity as quickly as possible once it has been perpetrated and, therefore, when fraud prevention has failed. In practice, fraud detection must be used continuously, since the system is unaware of when fraud prevention has failed. In addition fraud detection systems must evolve because once the detection methods has been discovered, criminals will adapt their strategies and circumvent it ([37]). Consequently an effective and efficient online detection system based on transaction monitoring needs to be a main point in the security policies of financial institutions.

Online banking fraud detection systems are centralised platforms located in the bank's data centre that collects heterogeneous information from the Internet Banking services. The system monitors and scrutinises each transaction, and compares its pattern with patterns built through past transactions history. In this way the software can score suspicious transactions *on-the-fly* and signal them to the bank's analyst for verification. All alerts generated from the detection system need to be manually investigated by the analyst of the bank, and this activity is very time consuming. As a result of this, the number of alerts should be kept at a level such that it can be handled by the available number of investigators and fraud analysts.

Since most customers rarely check their online banking history regularly and therefore are not able to discover and report fraud transactions immediately after their occurrences ([122]) frauds detection platforms are expected to continuously monitor transactions and to make almost instant frauds detection because the cost of a missing or a delayed detection is very high.

As a consequence the processing of the datasets by the fraud detection system requires fast and efficient machine learning and data mining algorithms and techniques.

Another problem in automated fraud detection system is the fact that one can never be completely sure if a transaction is really fraudulent or if it is a false positive. In fact, fraudsters make illicit transactions as similar as possible to regular transactions to prevent their discovery ([96]). Therefore it is nearly impossible to handle the fraud detection without human interaction.

Fraud detection mechanisms can make a good ranking of suspicious transactions, but the results normally have to be reviewed by a specialist nonetheless. This means that fraud detection mechanisms can only give a hint about which transactions have to be investigated further. Hence it is very important to have rules on how to weight false positives and false negatives. False negatives are often more costly than false positives [96], because an undetected fraudulent transaction can cause much more loss than a wrongly signalled legitimate transaction. Therefore monetary factors are often introduced to measure the performance of fraud detection mechanisms.

Another potential weak point of the automated fraud detection mechanisms is that they are not capable of detecting fraudulent behavior for which they are not designed. If a fraudster finds a way to circumvent the detection system, this fraud can go undiscovered, even for years. As long as the mechanisms do not adapt to the new situation, frauds will most certainly go undetected. The trend in newer works is therefore to create mechanisms than can adapt or be adjusted to new situations easily. This is not only because the fraudsters tend to adapt to the new detection mechanisms, but because the legitimate behavior tends to shift over time, too. As a remark it is also very important for the developer of anti-fraud mechanisms to maintain the old mechanisms to prevent fraudsters to switch back or to deny new, inexperienced fraudsters the access to old flaws ([37]).

2.3.3 Problem of Cooperation in Fraud Detection

Unfortunately, most of the work done in fraud prevention and fraud detection is not open to the public. There are several reasons. One is that if the new findings would be accessible by anyone, potential fraudsters can inform themselves about the newest development and adapt even faster as they do at the moment. Another important reason why most of the research work is not published, is that they often use sensitive data from a particular company. For banks, these are real transactional data which are under the protection of the banking secrecy. Despite the lack of real data there is a common sense about which attributes are important for the development of a good fraud detection mechanism. These attributes are often dates, amounts of transferred financial values, locations involved in the transaction, the transactional history, the payment history and other information([96]). To compensate these non disclosure agreements and the lack of exchange in

this sector, most studies are using artificially generated data ([57]). As with the experience of years of work, this data is nearly as realistic as real data would be and can therefore easily be used for further development and investigations. There are studies [32] with the topic of artificially generated data compared to real data and they state that it is a legitimate approach to generate synthetic data to train and implement new fraud detection mechanisms. However the results may vary when models built with simulated data are applied to real data.

The above characteristics make detection very difficult when applied to online banking fraud and presents several major challenges to the research: extremely imbalanced data, big data, model efficiency in dealing with complex data, dynamic data mining, pattern mining with limited or no labels, and discriminant analysis of data without clear differentiation.

2.4 State of Art in Internet Fraud Detection

In this section we cover the most relevant research works which specifically deal with topics related to the fraud detection.

First, we present techniques that focus the analysis of fraud detection on Internet Banking and its related field extracted from surveys (i.e., [43], [96], [37]), plus other relevant works.

2.4.1 Supervised Approach

The first category is the supervised approach on labeled data, and it is one of the most widely used approaches in this field of research and can be considered the typical data mining approach.

Various algorithms like Neural Network (e.g., [39], [60], [117], [100], [26]), Decision Tree (e.g., [95], [90]), case-based reasoning, regression methods, SVM ([124]), Rule-based Expert Systems ([109]), Bayesian Network ([41]), Contrast sets ([122], [33]), and Random Forests ([65]) are utilized here.

The ultimate goal is to distinguish fraudulent data from regular data. These approaches perform well in many classification applications, including fraud detection applications, even in certain class-imbalanced scenarios (e.g., [122], [26], [39], [100], [129]). As soon as this model is generated and applied to new data, it should be able to determine which part of the new data is fraudulent. Neural networks have been successfully adopted in all kinds of fraud detection scenarios and are believed to be a stable model.

The problem with supervised methods is that labeled data is difficult to come by. In fact, in the Internet Banking fraud detection field, labels are rarely provided. In addition some algorithms can only process certain types of attribute and some are too slow when applied to live data.

2.4.2 Unsupervised Approach

Unsupervised approaches such as link analysis, graph mining, HMM (e.g., [73], [89]), SOM ([100]), and Clustering (e.g., [66], [28], [125]) are often used to detect outliers or spikes when the underlying dataset is unlabeled. Such approaches do not use label information and have the advantage of detecting novel attacks, but usually have an overall accuracy lower than that of supervised approaches.

2.4.3 Semisupervised Approach

The third category consists of the semi-supervised approaches with only non-fraudulent data. In other words they assume that the training set consists only of correctly labeled benign instances.

Approaches in this category (e.g., [93]), try to identify fraudulent behavior by searching for instances that deviate from the learned model. This deviance then fires an alert that indicates a potential fraudulent behavior. These forms of machine learning are especially useful for credit card transactions, Internet Banking, and in the communication sector, where long term data is available.

2.4.4 Hybrid Approaches

Hybrid approaches use the same algorithms with the difference that more than one algorithm is used to generate the final rule set. Normally, these algorithms are used in a sequential way (e.g., [122], [108]). These approaches are used especially in the telecommunication sector.

2.4.5 Biological Approaches

Another approach we have identified is represented by methods from the bilogical sector (e.g., [113], [38]) that try to counter frauds using a system called AIS (Axon initial Segment) for detection. AIS try to approach the problem in a biological way, with negative and positive selection, training an artificial network to detect anomalies in transactions. They are usually used for detecting credit card fraud.

2.4.6 Statistical Approaches

Other approaches (see [37]) use statistical tools for fraud detection. They are quite diverse, since data from different applications can vary in both size and type. Such tools are essentially based on comparing the observed data with expected values wich, in turn, can be derived in various ways depending on the context. They may be represented by numerical and graphical

models of some aspects of the behavior but there are also more complex representations of multivariate behavior profiles.

Such behavior profiles may be based on past behavior of the system being studied (e.g., the way a bank account has been previously used) or be extrapolated from other similar systems. As [96] states, the possibilities of a visual fraud detection system seem to be ignored by most of the fraud detection community, as only very few papers use it. Statistical anomaly detection techniques (see [43]) are based on the following key assumption: normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model. Statistical techniques fit a statistical model (usually for normal behavior) to the given data and then apply a statistical inference test to determine if an unseen instance belongs to this model or not. Instances that have a low probability of being generated from the learned model, based on the applied test statistic, are declared as anomalies. Both parametric as well as non-parametric techniques have been applied to fit a statistical model. While parametric techniques assume the knowledge of the underlying distribution and estimate the parameters from the given data, non parametric techniques do not generally assume knowledge of the underlying distribution.

2.4.6.1 Histogram-Based

Histogram based approaches ([43]) can be considered the simplest non parametric statistical technique that use histograms to maintain a profile of the normal data. Such techniques are also referred to as frequency-based or counting-based. Histogram based techniques are particularly popular in the intrusion detection community (e.g., [52]) and fraud detection (e.g., [58]), since the behavior of the data is governed by certain profiles (user or software or system) that can be efficiently captured using the histogram model.

A basic histogram-based approach anomaly detection technique for univariate data consists of two steps. The first step involves building a histogram based on the different values taken by that feature in the training data. In the second step, the technique checks if a test instance falls in any of the bins of the histogram. If it does, the test instance is normal, otherwise it is anomalous. A variant of the basic histogram-based technique is to assign an anomaly score to each test instance based on the height (frequency) of the bin in which it falls.

The size of the bin used when building the histogram is key for anomaly detection. If the bins are small, many normal test instances will fall in empty or rare bins, resulting in a high false alarm rate. If the bins are large, many anomalous test instances will fall in frequent bins, resulting in a high false negative rate. Thus a key challenge for histogram-based techniques is to

determine an optimal size for the bins to construct the histogram that maintains a low false alarm rate and a low false negative rate.

Histogram based techniques require normal data to build the histograms. Some techniques even construct histograms for the anomalies, if labels for anomalous instances are available. For multivariate data, a basic technique is to construct attribute-wise histograms. In other words it builds a histogram for each feature. During testing, for each test instance, the anomaly score for each attribute value of the test instance is calculated as the height of the bin that contains the attribute value. The per-attribute anomaly scores are aggregated to obtain an overall anomaly score for the test instance. The basic histogram-based technique for multivariate data has been applied to system call intrusion detection, network intrusion detection, fraud detection, damage detection in structures, detecting Web-based attacks, and anomalous topic detection in text data (see [43]).

The key shortcoming of such techniques is that for multivariate data they are not able to capture the interactions between different attributes. For example, an anomaly might have attribute values that are individually very frequent, but whose combination is very rare. An attribute-wise histogram-based technique would not be able to detect such anomalies.

2.4.7 Credit card fraud detection

Credit card fraud can be divided into two types: offline fraud and online fraud.

Offline fraud is committed by using a stolen physical card. In most cases, the institution issuing the card can lock it before it is used in a fraudulent manner, if the theft is discovered quickly enough.

Online fraud is committed via web, phone shopping, or cardholder-not-present scenarios. With the increase of e-commence, online credit card transaction frauds are increasing likewise. Compared to online banking fraud detection, there are many available research discussions and solutions for credit card fraud detection (e.g., [113], [26], [124], [109], [111], [73], [90]). Most of the work on preventing and detecting credit card fraud has been carried out with neural networks (see [43, 77]).

Machine learning, adaptive pattern recognition, neural networks, HMM, SOM, and statistical modeling are employed to develop predictive models to provide a measure of certainty about whether a particular transaction is fraudulent or not.

[26] and [100] feature a neural network trained with the past data of a particular customer and uses the network to process current spending patterns to detect possible anomalies. [109] extracts a set of fuzzy association rules from a data set containing genuine and fraudulent transactions made with credit cards. [113] develops an AIS that generates normal memory cells using each user's transaction records and fraud memory cells are gen-

erated based on all fraudulent records. [90] use a decision tree learning algorithm called Very Fast Decision Tree, which scans real credit card transaction data as a data stream and use decision trees to detect frauds.

There are also some unsupervised methods, such HMM (e.g., [73]) and clustering (e.g., [111]), which targets unlabeled data sets.

The problem with most of the above approaches is that they require labeled data for both genuine as well as fraudulent transactions to train the classifiers. In fact getting real-world fraud data is one of the biggest problems associated with credit card fraud detection. Also, these approaches cannot detect new kinds of fraud for which labeled data is not available. They are not suitable for online banking because of the diversity of online banking customers' activities.

2.4.8 Fraud detection in online banking

As mentioned before there are very few papers about fraud detection in online banking.

Most of them concern fraud prevention, which uses efficient security measures to prevent fraudulent financial transactions performed by unauthorized users and to ensure transaction integrity.

[25] proposed an online banking fraud detection system for offline processing. Another system presented in [108] works well online but needs a component that must be downloaded and installed in the client device, which is inconvenient for deployment.

A supervised direction that emerged recently scrutinizes the difference between fraudulent and genuine behavior, and develops corresponding approaches for mining contrast patterns, for instance, contrast sets (e.g., [33]) and emerging patterns (e.g., [49]). According to the research in [121], contrast pattern mining is an NP hard problem, the time cost is expensive, especially when the number of attributes is large, and the threshold of minimal detection rate is small. This indicates that this methods does not perform efficiently in the online banking scenario. Another similar approach presented in [122] tries to overcome this shortcomings and treats events at different time points as dependent. It considers the information incorporated in event sequences in each transaction for differentiating fraudulent behavior from genuine behavior. In practice it introduces a hybrid model, combining contrast pattern with a neural network to increase its statistic modeling and reduce the number of "false" rejections.

2.4.9 Smartsifter

In our opinion, one of the most promising research directions is represented by the SmartSifter algorithm [126]. The approach of SmartSifter is as follows:

1. SmartSifter uses a probabilistic model as a representation of an underlying mechanism for data-generation. The model takes a hierarchical structure. A histogram density with a number of cells is used to represent a probability density over the domain of categorical variables. For each cell a finite mixture model is used to represent the probability density over the domain of continuous variables.

2. Every time a datum is input, SmartSifter employs an on-line learning algorithm to update the model. For the categorical domain a Sequentially Discounting Laplace Estimation algorithm is applied for learning the histogram density. For the continuous domain a Sequentially Discounting Expectation and Maximizing algorithm is applied for learning the finite mixture model underlying the distribution of the features under analysis. The most important feature of these two algorithms is that they gradually discount the effect of past examples in the on-line process.

3. SmartSifter assigns a score to each input datum on the basis of the learned model, measuring the change to the model after learning. A high score indicates a high possibility that the datum is an outlier.

In particular, SmartSifter shows the necessity to build systems adaptive to non-stationary sources. In the paper under analysis this is done by a discounting algorithm that learns from the source and forgets out-of-date statistics of the data using a decay factor. Another main point of the approach is represented by the score. It is calculated with a clear statistical/information-theoretic meaning. In fact it measures the change in the distribution learned from the data before and after the new datum is incorporated. This highlights the need of scores that do not just indicate if a transaction is an anomaly, but can be understood by the analyst. Another highlighted key point is the computational complexity. SmartSifter computes the score for each datum with a linear complexity in the number of parameters of the model and cubic in the number of variables. The last key aspect of SmartSifter is that it can deal with both categorical and continuous variables in an almost semi-automated way. We believe that this characteristics should be at the basis of every detection system that works in the Internet Banking field.

However this approach also presents some shortcomings. The first shortcoming we have detect is represented by the computational and spatial complexity. If from the one hand this complexity is less than the complexity of other methods, on the other hand it makes SmartSifter not applicable to real Online Banking data.

The second shortcoming is indeed the scarce readability of the model itself. In fact the score simply indicates how much a particular datum is anomalous, without giving the indication of how each attribute contributes

to the score or, in other words, why this particular datum has been detected as an outlier. These information are necessary to the analyst. In fact the model built by SmartSifter is complex, due to the multivariate data distribution that considers the relations between each attribute.

2.5 Open problems and research challenges

As mentioned in the previous section, different approaches have different advantages, but no single existing method can solve the online banking fraud detection problem easily. Hence it is necessary to analyze the main problems the described approaches have to cope with.

One of the main problems is that they are not designed on a study of real transactions data. Real transactions data has many peculiarities (e.g. very skewed dataset, huge attribute cardinality) which requires several adjustments to the typical statistical and data mining methods used in the outlier detection field. An approach not tested on real data may be not applicable to real scenarios and this seems to be confirmed by the fact that only few of the proposed methods were actually implemented in a productive system. This is also confirmed by [122] which states that classic methods demonstrates poor performance when directly applied to online banking fraud detection.

A second strong drawback of existing approaches is that nearly none of them incorporates temporal data and spatial information for fraud detection. In transactional data, temporal and spatial information are almost always available. This information can, together with other information, give a good hint for fraudulent or non fraudulent behavior. For example a lot of transactions from one account in a given EU country to an account in a non-EU country can be suspicious.

A third issue is that it is difficult to compare performance between state-of-the-art approaches, because they all use different metrics due to the great variety of methods adopted. This aspect is aggravated by the fact that performances depend on the dataset features and on their distributions.

A fourth criticism can be made on the fact that the majority of the described approaches build black box models and thus, are not easily interpretable: the system communicates only a score indicating the anomaly, without any additional information for the bank analyst. As written before, the final output of a transaction monitoring system goes to the analyst, who must investigate over the indicated transaction. This is a time consuming activity.

Another drawback of previous work is that nearly none of them handle the problem of dealing with the scarcity of data that might not be enough to train an anomaly detection system in a reasonable time frame. The only work we have found is [104], in which they propose an approach independent from the contest which addresses local training data deficiencies by ex-

ploiting clustering techniques to construct a knowledge base of well-trained models. These models are then used to handle the case of an undertrained user.

A shortcoming of most of the described methods is their complexity. As stated before, fraud detection is a time critical matter, especially when it comes to adapting to new fraudulent techniques. This is because as long as the detection mechanism do not adapt, the fraudulent transactions are not recognized. Complex techniques can indeed be more accurate, but often they are slower than the less complex ones.

Finally, some of the approaches seen before build profiles on global models of frauds or fraudulent behavior. These approaches generally have a lower accuracy than approaches that build profiles that model each user's behavior, in particular if a country, region or bank has a very specific type of customers with atypical behavior.

3 Evolutions of Banking Trojans

The Internet has become the infrastructure of choice for storing, transmitting and using sensitive personal and business information. A large and diverse population of users accesses online banking services, or performs different kinds of electronic financial transactions. Unsurprisingly, endpoint devices such as computers, mobile phones and tablets have become easy targets for cyber criminals, who infect devices with malware that is built to steal sensitive data or perform fraudulent monetary transactions without the owner's consent.

A recent study of the Russian underground market of cyber criminals [61] estimated a USD 2.3 billion market, that is 18% of the estimated total USD 12.5 billion worldwide cybercrime figure in 2011 according to [23]. In this market, malicious goods are a "service" with a price tag: from encryption to servers, from distributed denial-of-service attacks to spamming. A spam campaign is particularly cheap, costing down to USD 10 per million of emails. Anybody can easily buy a customized trojan, or a malware-building toolkit to create a customized sample. Malware authors and their "affiliate" employees even offer paid support and customizations, or sell advanced configuration files that the customers can include in their builds, for instance to add new functionality, or to target the users of a specific website. The customer can pay on a per-installation basis, with prices depending on the targeted country: 1,000 infected Russian users, for instance, cost approximately USD 100. This malware-as-a-service phenomenon [62] is alarming, as it turns botnets into a commodity, and allows traditional crime gangs to enter the cyberfraud landscape. Unsurprisingly, online banking fraud is one of the fastest growing segments of cybercrime, amounting to just below USD 1 billion [23].

3.1 Zeus: the most successful example of banking trojan

Among these menaces, information-stealing trojans are a growing [86], sophisticated threat. The most famous examples are ZeuS and SpyEye. Since its first appearance in 2007, Zeus has grown into one of the most popular families of credential-stealing trojans. Due to its popularity, previous versions of Zeus have been extensively investigated by the security community [56, 123].

Information-stealing trojans allow a malware operator to intercept credentials such as usernames, passwords, and second factors of authentication (e.g., PINs or token-generated codes). These trojans are also referred to as "banking trojans", because they are often used to steal banking credentials when the victim is using an online banking service. As we detail in Section 3.4.1.1, the typical information stealer implements the interception mechanism through injection modules. An injection module, codenamed "WebInject" in the case of ZeuS and SpyEye, manipulates and injects arbitrary content into the data stream transmitted between an HTTP(S) server and the user browser. The injection modules are placed between the rendering engine of the browser and the network-level libraries. Thus, they are able to circumvent any form of transmission encryption such as SSL, as we describe in Section 3.4.1.2. A comprehensive list of Trojan families used to conduct Man-in-the-Browser (MitB) attacks is presented in Table 3.1

The internals of the first two versions of Zeus, which are based on centralized command and control (C2) servers, are well understood, and C2 servers used by these variants are routinely tracked and blocked. [1]

In May 2011, the source code of the second centralized version of Zeus was leaked. This has led to the development of several centralized trojans based on Zeus, such as ICE IX [118], and the more successful Citadel [112]. In September 2011, a peer-to-peer (P2P) mutation of centralized Zeus appeared, known as *P2P Zeus* or *GameOver*. Due to its lack of centralized C2 servers, P2P Zeus is not susceptible to traditional anti-Zeus countermeasures, and is much more resilient against takedown efforts than centralized Zeus variants. In this section, we perform a detailed analysis of the P2P Zeus protocol to highlight how it achieves its resilience. Our insights also shed light on the resilience potential of peer-to-peer botnets in general.

Centralized Zeus variants are sold as builder kits in the underground community, allowing each user to build a private Zeus botnet.

[1] http://zeustracker.abuse.ch

Adramax	Murofet (Licat)
Ainslot	Neloweg
Ares	Nimkey
Banbra (Dadobra, Nabload)	Nuklus (Apophis)
Bancos	OddJob
BankDiv (Banker.BWB)	Origami
Bankolimb (NetHell, Limbo)	Papras (Snifula)
BankPatch (Patcher)	Pophot
Bradesco	PowerGrabber
Bjlog	Qakbot (Qbot)
Briz (VisualBreez)	QQLogger
Bugat (Feodo, Cridex)	QQShou
Carberp (Syscron)	Ramnit (DesktopLayer)
Chekafev	Ruftar
Cimuz (Bzud, Metafisher, Abwiz, Agent-DQ)	Shylock
Citadel	SilentBanker
Clampi	Silon
DBJP	Sinowal (Wsnpoem, Anserin, AudioVideo)
Dumador (Dumarin, Dumaru)	Snatch (Gozi)
Dybalom	Specbot
Eurosol	SpyEye
Family18	Spyforms
Fingotok	SunSpot
Gameover (P2P ZeuS)	Tatanga (Gataka, Hermes)
Goldun (Haxdoor, Nuclear Grabber)	Tepfer
Ice IX	Tigger
Kykymber	Tilon
LdPinch	Tiny Banker (Tinba, Zusy)
Lego	Torpig (Xorpig, Mebroot)
Lenin	URLZone
Leprechaun	Usteal
Lmir	Vkont
Malintent	Wemon
Mimicker	ZeuS (Zbot)

Table 3.1: A summary of MITB trojan families

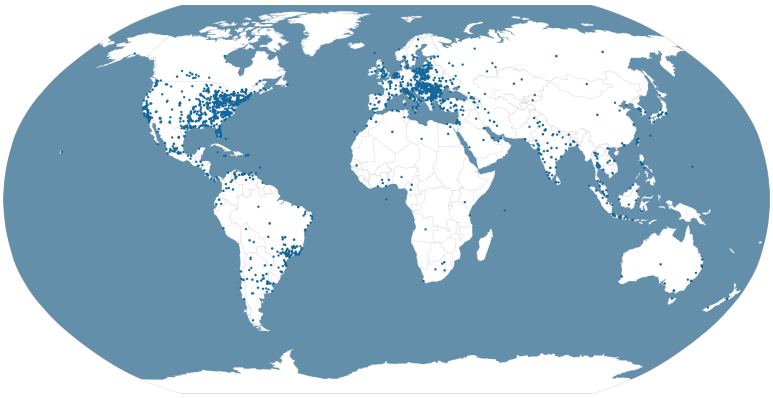


Figure 3.1: Geographical distribution of externally reachable Zeus peers.

3.2 Peer to Peer Network Topology and challenges

In this Section, we analyze in detail a very damaging and resilient example of malware: the peer-to-peer version of the Zeus banking trojan. We are particularly interested in resilience, as it indicates how advanced modern malware has become and how hard it will be to dismantle it – now and even more so in the near future.

The main P2P network is divided into several virtual *sub-botnets* by a hardcoded sub-botnet identifier in each bot binary. While the Zeus P2P network is maintained and periodically updated as a whole, the sub-botnets are independently controlled by several botmasters. Bot enumeration results from our previous work indicate that the Zeus P2P network contains at least 200.000 bots [105]. The number is fairly stable. In other words, if the botnet loses, say, 50.000 nodes due to a sinkholing attempt, it will grow again to roughly the same size (and no more). The geographical distribution of the externally reachable peers is shown in Figure 3.1.

The Zeus P2P network serves two main purposes. First, bots exchange binary and configuration updates with each other. Second, bots exchange lists of *proxy bots*, which are designated bots where stolen data can be dropped and commands can be retrieved. Additionally, bots exchange neighbor lists (*peer lists*) with each other to maintain a coherent network. As a backup channel, P2P Zeus also uses a Domain Name Generation Algorithm (DGA) [29], in case contact with the regular P2P network is lost.

Our contributions in this deliverable on the subject of P2P Zeus networks are:

1. We reverse engineered the entire Zeus P2P protocol and topology, and will highlight the features that increase the botnet's resilience to takedown attempts.

2. We show that P2P Zeus has evolved into a complex bot with attack capabilities that go beyond typical banking trojans. Particularly, we find that P2P Zeus is used for activities as diverse as DDoS attacks, malware dropping, Bitcoin theft, and theft of Skype and banking credentials.

3. Reports from academia and industry have long warned of the high resilience potential of peer-to-peer botnets [128, 47, 48, 127, 69]. Through our analysis of the communication protocol and resilience mechanisms of P2P Zeus, we show that highly resilient P2P botnets are now a very real threat.

3.2.1 Structure of the Zeus P2P network

The Zeus network is organized into three disjoint layers, as shown in Figure 3.2. At the bottom of the hierarchy is the *P2P layer*, which contains the bots. Periodically, a subset of the bots is assigned the status of *proxy bot*. This appears to be done manually by the botmasters, by pushing a cryptographically signed *proxy announcement* message into the network. The details of this mechanism are explained in Section 3.2.2. The proxy bots are used by harvester bots to fetch commands and drop stolen data. Aside from their special function, proxy bots behave like harvester bots.

The proxy bots act as intermediaries between the P2P layer and a higher layer, which we call the *C2 proxy layer*. The C2 proxy layer contains several dedicated HTTP servers (not bots), which form an additional layer between the proxy bots and the true root of the C2 communication. Periodically, the proxy bots interact with the C2 proxy layer to update their command repository, and to forward the stolen data collected from the bots upward in the hierarchy.

Finally, at the top of the hierarchy is the *C2 layer*, which is the source of commands and the final destination of stolen data. Commands propagate downward from the C2 layer, through the C2 proxy layer to the proxy bots, where they are fetched by harvester bots. Similarly, data stolen by harvester bots is collected by the proxy bots, and periodically propagated up until it ultimately reaches the C2 layer.

As mentioned in Section 3.1, the main P2P network is divided into several virtual *sub-botnets* by a hardcoded sub-botnet identifier in each bot binary. Since each of these sub-botnets is independently controlled, the C2 layer may contain multiple command sources and data sinks.

3.2.2 The zeus P2P Protocol

This section describes our analysis results on the Zeus P2P communication protocol. The results are based on Zeus variants we tracked between February 2012 and July 2013. In that time, several changes were made to

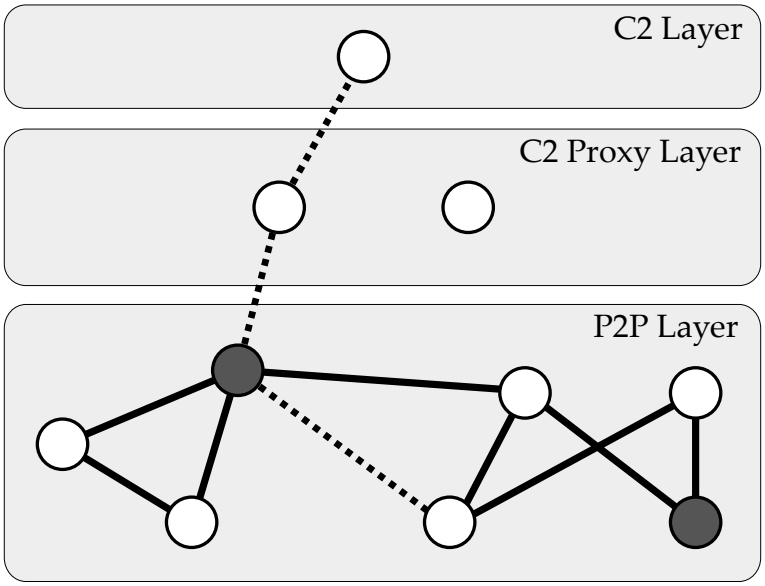


Figure 3.2: Topology of P2P Zeus. Shaded nodes represent proxy bots. The dotted line shows the information flow between a harvester bot and the C2 layer.

the protocol by the Zeus authors. The results presented here apply to all recent P2P Zeus versions, except where noted differently.

We first provide a high level overview of the Zeus P2P protocol in Section 3.2.2.1. Next, we describe the encryption used in Zeus traffic in Section 3.2.2.2. Sections 3.2.2.3 and 3.2.2.4 provide a detailed overview of the Zeus message structure. Finally, Section 3.2.3 describes in detail how the Zeus P2P protocol operates.

3.2.2.1 Overview

As mentioned in Section 3.1, the Zeus P2P network's main functions are (1) to facilitate the exchange of binary and configuration updates among bots, and (2) to propagate lists of *proxy bots*. Most normal communication between bots is based on UDP. The exceptions are Command and Control (C2) communication between harvester bots and proxy bots, and binary/-configuration update exchanges, both of which are TCP-based.

Bootstrapping onto the network is achieved through a hardcoded bootstrap peer list. This list contains the IP addresses, ports and unique identifiers of up to 50 Zeus bots. Zeus ports range between 1024 and 10000 in versions after June 2013, and between 10000 and 30000 in older versions. Unique identifiers are 20 bytes long and are generated at infection time by taking a SHA-1 hash over the Windows *ComputerName* and the Volume ID

of the first hard-drive. These unique identifiers are used to keep contact information for bots with dynamic IPs up-to-date.

Network coherence is maintained through a push-/pull-based peer list exchange mechanism. Zeus generally prefers to push peer list updates; when a bot receives a message from another bot, it adds this other bot to its local peer list if the list contains less than 50 peers. Bots in desperate need of new peers can also actively request them. In principle, the peer pushing mechanism facilitates peer list poisoning attacks against Zeus. However, as we will see in Sections 3.2.2.2, 3.2.3.1 and 3.3, Zeus includes several resilience measures which severely complicate poisoning attacks.

Zeus bots check the responsiveness of their neighbors every 30 minutes. Each neighbor is contacted in turn, and given 5 opportunities to reply. If a neighbor does not reply within 5 retries, it is deemed unresponsive, and is discarded from the peer list. During this verification round, every neighbor is asked for its current binary and configuration file version numbers. If a neighbor has an update available, the probing bot spawns a new thread to download the update. Updates are signed using RSA-2048, and are applied *after* the bot has checked that the update's embedded version number is higher than its current version. Thus, it is impossible to force bots to "update" to older versions.

The neighbor verification round is also used to pull peer list updates if necessary. If the probing bot's peer list contains less than 25 peers, it asks each of its neighbors for a list of new neighbors. The returned peer lists can contain up to 10 peers. The returned peers are selected by minimal Kademlia-like XOR distance to the requesting bot's identifier [88]. However, we note that the Zeus P2P network is *not* a Distributed Hash Table, and apart from this XOR metric the protocol bears no resemblance to Kademlia.

In case a Zeus bot finds all of its neighbors to be unresponsive, it attempts to re-bootstrap onto the network by contacting the peers in its hard-coded peer list. If this also fails, the bot switches to a DGA backup channel, which can be used to retrieve a fresh, RSA-2048 signed, peer list. Additionally, in recent variants of Zeus, the DGA channel is also contacted if a bot is unable to retrieve updates for a week or longer. This is a very important resilience feature, as it allows the botnet to recover from peer list poisoning attacks. The DGA mechanism is described in more detail in Section 3.3.

As mentioned, one of the most important functions of the Zeus P2P network is to propagate lists of proxy bots. These proxy bots are periodically selected from the general bot population, and are contacted by bots to fetch commands from and drop stolen data to. Like the peer list exchange mechanism, the proxy list mechanism is also push-/pull-based. When a new proxy bot is appointed by the botmasters, an RSA-2048 signed push message is disseminated through the network to announce it.

Bots are commanded in two ways. First, harvester bots can contact proxy bots to retrieve commands. Second, configuration file updates can also be used to convey commands to the bots.

3.2.2.2 Encryption

Until recently, bot traffic was encrypted using a rolling XOR algorithm, known as "visual encryption" from centralized Zeus [123], which encrypts each byte by XORing it with the preceding byte. Since June 2013, Zeus uses RC4 instead of the XOR algorithm, using the recipient's bot identifier as the key. Rogue bots used by analysts to infiltrate the network typically use continuously changing bot identifiers to avoid detection [105]. The new RC4 encryption is a problem, because a rogue bot may not always know under which identifier it is known to other bots, thus preventing it from decrypting messages it receives. In addition, RC4 increases the load on botnet detection systems which rely on decrypting C2 traffic [106].

Zeus uses RSA-2048 to sign sensitive messages originating from the botmasters, such as updates and proxy announcements. In all P2P Zeus variants we studied, update exchanges and C2 messages feature RC4 encryption over an XOR encryption layer. For these messages, either the identifier of the receiving bot or a hardcoded value is used as the RC4 key, depending on the message type.

3.2.2.3 Message Structure

This section describes the structure of Zeus network messages. Zeus messages vary in size, but have a minimum length of 44 bytes. The first 44 bytes of each message form a header, while the remaining bytes form a payload concatenated with padding bytes. The Zeus message structure is illustrated in Figure 3.3. The following message structure diagrams are to scale. Shaded areas do not represent part of the message structure itself, but serve to align the fields in the figures.

3.2.2.3.1 rnd (random) In Zeus versions which use the XOR encryption, this byte is set to a random value. This is done to avoid leaking information, since the XOR encryption leaves the first byte in plaintext. In Zeus versions which use RC4 for message encryption this byte is set to match the first byte of the session ID, so that it can be used to confirm that packet decryption was successful. Backward compatibility with older bots is achieved by falling back to the XOR encryption if RC4 decryption fails.

3.2.2.3.2 TTL (time to live) The TTL field is usually unused, in which case it is set to a random value, or to the second byte of the session ID

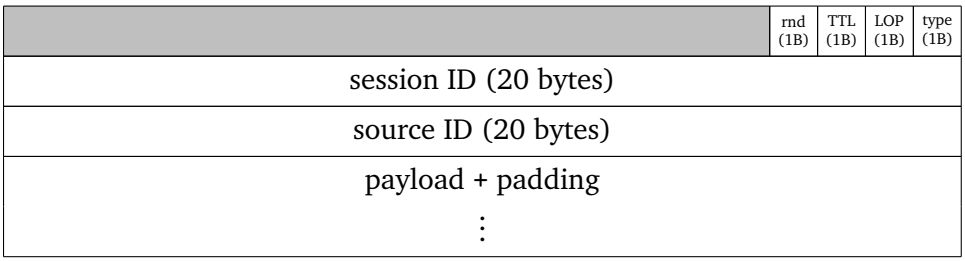


Figure 3.3: The Zeus message structure.

for variants using RC4 encryption. However, for certain message types, this field serves as a time to live counter. A bot receiving a message using the TTL field forwards it with a decremented TTL. This continues iteratively until the TTL reaches zero.

3.2.2.3.3 LOP (length of padding) Zeus messages end with a random amount of padding bytes. This is most likely done to confuse signature-based intrusion detection systems. The length of padding field indicates the number of padding bytes appended to a message.

3.2.2.3.4 type This field indicates the type of the message. The message type is used to determine the structure of the payload, and in certain cases the meaning of some of the header fields, such as the TTL field. Valid Zeus message types are described in Section 3.2.2.4.

3.2.2.3.5 session ID When a Zeus bot sends a request to another bot, it includes a random session ID in the request header. The corresponding reply will include the same session ID, and incoming replies with unexpected session ID values are discarded. This makes it more difficult for attackers to blindly spoof Zeus messages.

3.2.2.3.6 source ID This field contains the 20 byte bot identifier of the sending bot. The source ID field facilitates the push-based peer list update mechanism, where a bot receiving a message adds the sender of the message to its peer list in case the peer list contains less than 50 peers.

3.2.2.3.7 payload This is a variable-length field which contains a payload that depends on the message type. The structures of relevant message payload types are described in detail in Section 3.2.2.4.

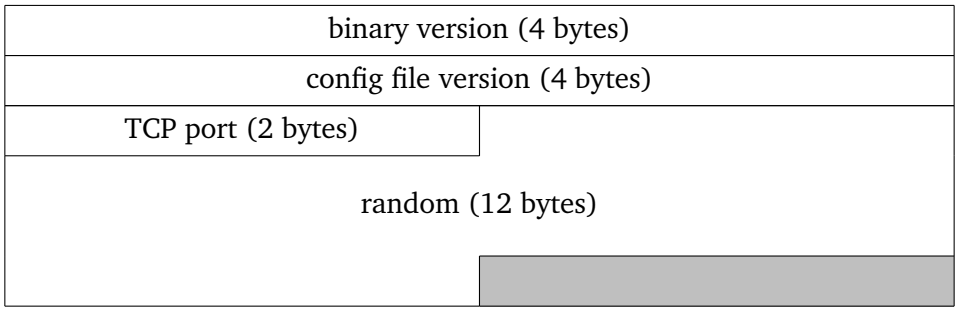


Figure 3.4: Version reply payload (22 bytes).

3.2.2.3.8 padding This field contains a random number of random (non-zero) padding bytes. The number of padding bytes is specified in the length of padding field in the message header.

3.2.2.4 Payload Structure

In this section, we describe the structure and usage of the most relevant Zeus message types. Each of these message types is communicated over UDP, except for C2 messages and updates, which are exchanged over a TCP connection.

3.2.2.4.1 Version request (type `0x00`) Version request messages are used to request a bot's current binary and configuration file version numbers. These messages usually contain no payload, but may contain a payload consisting of a little endian integer with value 1, followed by 4 random bytes. Such a payload serves as a marker to indicate that the requesting peer wants to receive a type `0x06` proxy reply message (see Section 3.2.2.4.7).

3.2.2.4.2 Version reply (type `0x01`) A version reply contains the version numbers of the binary and configuration files of the sender. The binary version indicates the sender's Zeus version, while the configuration file version indicates the sender's configuration file version. A TCP port is also sent, which may be contacted to download the updates via TCP, although some Zeus variants also support using UDP for this (see Sections 3.2.2.4.5 and 3.2.2.4.6). Version replies end with 12 random bytes. The reply structure is shown in Figure 3.4.

3.2.2.4.3 Peer list request (type `0x02`) Peer list requests (Figure 3.5) are used to request new peers from other bots. Zeus only sends active peer list requests if its peer list is becoming critically short (less than 25 peers). Otherwise, bots typically rely on storing the senders of incoming requests.

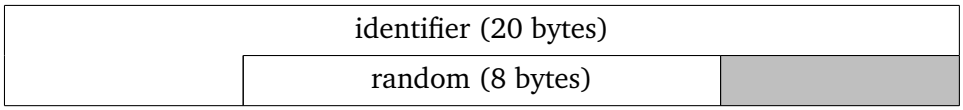


Figure 3.5: Peer list request payload (28 bytes).

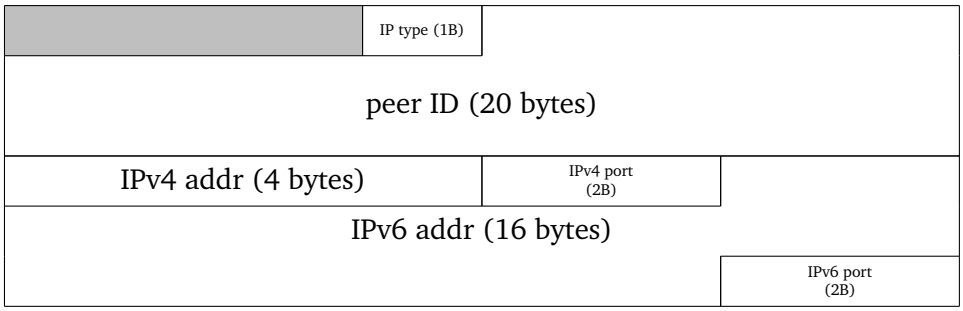


Figure 3.6: Peer struct (45 bytes).

The payload of a peer list request consists of a 20 byte identifier, followed by 8 random bytes. The responding peer will return the peers from its own peer list that are closest to the requested identifier.

3.2.2.4.4 Peer list reply (type `0x03`) Peer list replies contain 10 peers from the responding peer's peer list which are closest to the requested identifier. If the responding peer knows fewer than 10 peers, then as many peers as possible (potentially zero) are returned, and any remaining peer slots are zeroed out. For each returned peer, the payload format is as shown in Figure 3.6. Zeus supports both IPv4 and IPv6, but in practice we have observed very few IPv6 peers. The IP type field indicates whether the peer is reachable via IPv4 (set to 0) or IPv6 (set to 2). The remaining fields contain the peer's identifier, IP address and UDP port. Any unused fields are randomized.

3.2.2.4.5 Data request (type `0x04/0x68/0x6A`) A UDP data request payload, shown in Figure 3.7, starts with a single byte indicating the kind of requested data. This byte is set to 1 for a configuration file download, or to 2 for a binary update. The offset field indicates the word offset into the data to be transmitted and the size field specifies how many data bytes should be sent. TCP data requests consist of a message header with type `0x68` for a binary request, or type `0x6A` for a configuration request.

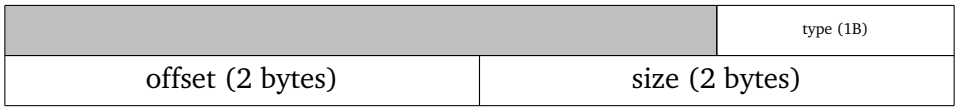


Figure 3.7: Data request payload (5 bytes).

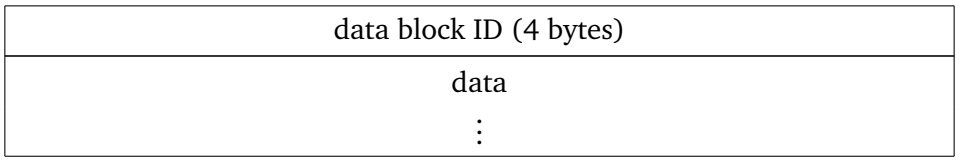


Figure 3.8: Data reply payload (length varies).

3.2.2.4.6 Data reply (type `0x05/0x64/0x66`) UDP data transfers (type `0x05`) are sent in chunks of 1360 bytes, until no more data is available. If a bot receives a data reply containing less than 1360 data bytes, it assumes that this is the last data block, and ends the download. If a data reply takes longer than 5 seconds to arrive, the download is aborted, and the maximum total size of any download is 10MB. These constraints mean that it is not possible to launch "tarpit" attacks, where bots are tied up by very slow and never ending downloads.

Each data reply (Figure 3.8) starts with a 4 byte randomly chosen file identifier, followed by the requested data. The transmitted files end with an RSA-2048 signature over the MD5 hash of the plaintext data, and are encrypted with RC4 using a hardcoded key on top of an XOR encryption layer. Before applying an update, Zeus checks that the version number contained in the update is strictly higher than its current version number. This means that it is not possible to make Zeus bots revert to older versions.

TCP data transfers start with a message header of type `0x64` for a binary update, or `0x66` for a configuration update, followed by the RC4 encrypted data.

3.2.2.4.7 Proxy reply (type `0x06`) Proxy replies return proxy bots in response to version requests carrying a proxy request marker. A proxy reply can contain up to 4 proxy bot entries, each of which is RSA-2048 signed. Each proxy entry is formatted as shown in Figure 3.9. The format is similar to that used in peer list replies, except that the IP type field is 4 bytes long, and there is an RSA signature at the end of each proxy entry.

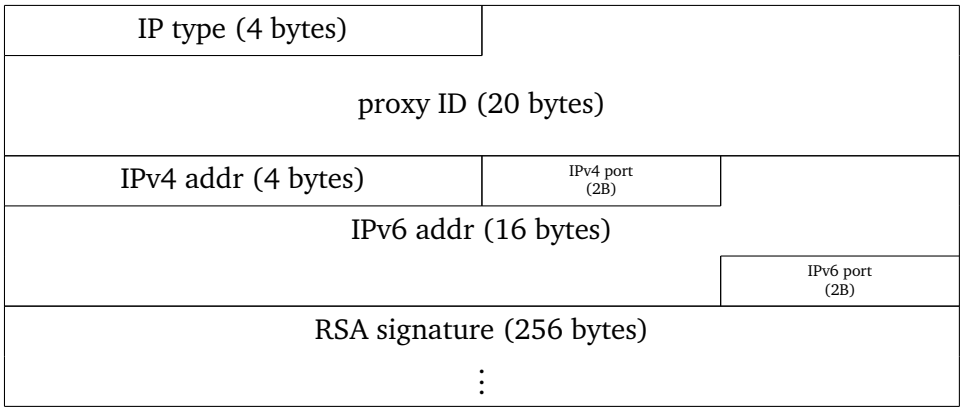


Figure 3.9: Proxy struct (304 bytes).

3.2.2.4.8 Proxy announcement (type `0x32`) Proxy announcements are similar to proxy replies, but are actively pushed through the Zeus network by bots which are appointed as proxies by the botmasters. Newly appointed proxy bots announce themselves to all their neighbors, which pass on the message to all their neighbors, and so on. This continues until the TTL field (Section 3.2.2.3) reaches zero. The TTL field has an initial value of 4 for proxy announcements. Thus, proxy announcements propagate very rapidly, although they cannot reach NATed bots directly. Proxy announcements contain a single proxy entry of the same format used in type `0x06` messages, as shown in Figure 3.9.

3.2.2.4.9 C2 message (type `0xCC`) Unlike most message types, C2 messages are only exchanged between harvester bots and proxy bots, and are exchanged over TCP. C2 messages are used as wrappers for HTTP messages. Because of this, we suspect that the communication between proxy bots and the C2 proxy layer is HTTP-based. The HTTP-based C2 protocol is heavily based on the C2 protocol used in centralized Zeus [34, 56]. An example C2 HTTP header for a command request is shown in Figure 3.10. The X-ID field specifies the sub-botnet for which a command is being requested.

The HTTP header is followed by an HTTP payload, which consists of several, optionally zlib-compressed, data fields. The payload begins with a header specifying the payload size and flags, and the number of data fields that follow. The payload header ends with an MD5 hash of the combined data fields, which is used to verify message integrity.

Each data field is XOR encrypted, and starts with a header specifying the field type, flags, and compressed and uncompressed sizes. The header is

```
POST /write HTTP/1.1
Host: default
Accept-Encoding:
Connection: close
Content-Length: 400
X-ID: 100
```

Figure 3.10: C2 HTTP header.

Type	Content
0x65	System name and volume ID
0x66	Bot identifier
0x67	Infecting spam campaign
0x6b	System timing information
0x77	Stolen data

Table 3.2: Typical C2 request fields.

followed by the actual data, the structure of which is dependent on the field type.

C2 request messages typically contain several fields describing status of and information about the requesting bot. Typical fields included in C2 requests are shown in Table 3.2. Note that the type numbers of data fields are completely independent from Zeus message type numbers.

The most important data field contained in a C2 response is the command field, which has type 0x01. It contains an MD5 hash used to verify integrity of the command, followed by the command itself in the form of an ASCII-string. Notable command strings are listed in Table 3.3.

As can be seen from Table 3.3, Zeus supports a diverse set of commands, which goes far beyond that of a typical banking trojan. The supported

Command	Meaning
user_execute	Execute file at URL
user_certs_get	Steal crypto certificates
user_cookies_get	Steal cookies
ddos_url	DDoS a given URL
user_homepage_set	Set homepage to URL
fs_pack_path	Upload local files to botmaster
bot_bc_add	Open VNC server

Table 3.3: Notable C2 command strings.

commands include dropping files, launching DDoS attacks, providing remote access to the botmasters, and stealing a plethora of credentials. Aside from banking credentials, we have observed Zeus stealing Skype and MSN database files, as well as Bitcoin wallets.

3.2.3 Communication Patterns

Each Zeus bot runs a passive thread, which listens for incoming requests, as well as an active thread, which periodically generates requests to keep the bot up-to-date and well-connected. We describe the behavior of each of these threads in turn.

3.2.3.1 Passive thread

Every Zeus bot listens for incoming messages in its passive thread. A Zeus bot receiving an incoming request attempts to handle this request as described in Section 3.2.2.4. The sender of any successfully handled request is considered for addition to the receiving bot's peer list. This is the main mechanism used by *externally reachable* Zeus bots to learn about neighbors, and it is also how new bots introduce themselves to the network. If the receiving bot has fewer than 50 neighbors, then it always adds the sender of the request to its peer list. Additionally, if the identifier of the sender is already present in the peer list, then both corresponding IP address and port are updated. This is done to accommodate senders with dynamic IPs and discard stale dynamic IPs. If the identifier of the sender is not yet known, but the peer list already contains 50 peers or more, then the sending peer is stored in a queue of peers to be considered for addition during the next neighbor verification round (see Section 3.2.3.2).

Before adding a new peer to the peer list, a number of sanity checks are performed. First, only peers which have a source port in the expected range are accepted. NATed bots may make it into the peer lists of other bots, if they happen to choose a port in the valid range. Additionally, only one IP address per /20 subnet may occur in a bot's peer list at once. This defeats peer list poisoning attempts which use IP ranges within the same subnet. Recent versions of Zeus also include an automatic blacklisting mechanism, which blacklists IPs that contact a bot too frequently in a specified time window. This mechanism further complicates efficient crawling and poisoning of the network.

When a type `0x32` proxy announcement arrives, its signature is first checked for validity. If the message passes the check, the TTL field is decremented and the message is forwarded to all known neighbors if the TTL is still positive. Furthermore, new proxy bots which pass verification are considered for addition to the receiving bot's proxy list. The proxy list is similar to the peer list, but is maintained separately. If the identifier of the

new proxy is already in the proxy list, then both corresponding IP address and port are updated. Otherwise, if a proxy list entry is found that is over 100 minutes older than the new proxy, this entry is overwritten with the new proxy (this is not done for type `0x06` proxy replies). In any other case, the new proxy is added to the end of the proxy list. Finally, the proxy list is truncated to its maximum length of 10 entries, effectively discarding the new proxy if the proxy list was already 10 entries long.

3.2.3.2 Active thread

The Zeus active communication pattern consists of a large loop which repeats every 30 minutes. The function of the active communication loop is to keep Zeus itself, as well as the peer list and proxy list, up to date.

In each iteration of the loop, Zeus queries each of its neighbors for their binary and configuration file versions. This step serves to keep the bot up to date, and to check each neighbor for responsiveness. If Zeus knows fewer than 4 proxy bots, it piggybacks a proxy request marker with each version request. Each peer is given 5 chances to respond to a version request. If a peer fails to answer within the maximum number of retries, Zeus checks if it has working Internet access by attempting to contact `www.google.com` or `www.bing.com`. If it does, the unresponsive peer is discarded. If the peer responded and is found to have an update available, the update is downloaded in a separate thread.

After version querying all peers in its peer list, Zeus proceeds to handle any pending peers which were queued from incoming requests (see Section 3.2.3.1). Pending peers are only handled if the peer list contains fewer than 50 peers, and the procedure is stopped as soon as the peer list reaches length 50. Each pending peer is sent a single version request, and is added to the peer list if it responds.

Finally, if the peer list contains fewer than 25 peers, the bot will actively send peer list requests to each of its neighbors until the peer list reaches a maximum size of 150 peers. This is only done once every 6 loop cycles (3 hours), and is an emergency measure to prevent the bot from becoming isolated. If, despite this effort, a bot does find itself isolated, it will attempt to recover connectivity by contacting its hardcoded bootstrap peer list. If this also fails, the bot will enter DGA mode, as further described in Section 3.3.

3.3 Domain Name Generation Algorithm

As mentioned in Section 3.2.2.1, Zeus contains a Domain Generation Algorithm, activated if all of a bot's neighbors are unresponsive, or the bot cannot fetch updates for a week. The DGA generates domains where Zeus can download a fresh RSA-2048 signed peer list. The DGA is a very potent

```
for(i = 0; i < 1000; i++) {
    S[0] = (year + 48) % 256;   S[1] = month;
    S[2] = 7 * (day / 7);       *(int*)&S[3] = i;

    /* convert hash to domain name */
    name = "";  hash = md5(S);
    for(j = 0; j < len(hash); j++) {
        c1 = (hash[j] & 0x1F) + 'a';
        c2 = (hash[j] / 8) + 'a';
        if(c1 != c2 && c1 <= 'z') name += c1;
        if(c1 != c2 && c2 <= 'z') name += c2;
    }

    /* select TLD for domain */
    if(i % 6 == 0) name += ".ru";
    else if(i % 5 != 0) {
        if(i & 0x03 == 0) name += ".info";
        else if(i % 3 != 0) {
            if((i % 256) & 0x01 != 0) name += ".com";
            else name += ".net";
        } else name += ".org";
    } else name += ".biz";

    domains[i] = name;
}
```

Figure 3.11: The P2P Zeus Domain Name Generation Algorithm.

backup mechanism, which makes long term poisoning or sinkholing attacks against Zeus very difficult [105].

3.3.1 Algorithm Details

The Zeus Domain Generation Algorithm generates 1000 unique domains per week. A bot entering the DGA starts at a random position in the current week's domain list and sequentially tries all domains until it finds a responsive domain. The DGA uses top-level domains taken from the set {`biz`, `com`, `info`, `net`, `org`, `ru`}. The Zeus DGA bears some resemblance to the DGA of Murofet, a malware known to be related to centralized Zeus [72].

The Zeus DGA is shown in C-like pseudocode in Figure 3.11. The code shown generates all 1000 domains for a given week. The generation of a domain name starts by taking the MD5 hash over the concatenation of (transformations of) the year, month, day, and domain index. The MD5 hash is then used to generate a domain name of at most 32 lower case alphabetic characters. Finally, the domain is completed by selecting one of the six top-level domains and concatenating it to the domain name.

3.4 Analyzing web injections

3.4.1 Information-stealing Trojans: Overview and Challenges

State-of-the-art malware is very sophisticated and the development industry is quite mature. [82] recently measured that trojans such as ZeuS and GenericTrojan are actively developed and maintained. Indeed, both malware families live in a complex environment with development kits, web-based administration panels, builders, automated distribution networks, and easy-to-use customization procedures. The most alarming consequence is that anyone can buy a malware builder from underground marketplaces and create a customized sample. Interestingly, cyber criminals also offer paid support and customizations, or sell advanced configuration files that the end users can include in their custom builds, for instance to extract information and credentials of specific (banking) websites. [82] also found an interesting development evolution, which indicates a need for forward-looking malware-analysis methods that are less dependent on the current or past characteristics of the malware.

3.4.1.1 WebInject Functionality

From hereinafter we use ZeuS and SpyEye as a study case. However, as detailed in Section 3.4.2, our approach does not rely on their implementation of the WebInject functionality.

Malware families that follow the same approach of ZeuS and SpyEye usually include data-stealing functionalities. For instance, since version 1.0.0, SpyEye features a so-called "FormGrabber" module, which can be arbitrarily configured to intercept the data that the victim types into (legitimate) websites' forms. This type of trojans are often referred to as "infostealers", in jargon. Unsurprisingly, the main goal of money-motivated criminals that rent or operate information-stealing campaigns is to retrieve valid, full credentials from infected systems. Online-banking websites credentials are among the most targeted ones. Typically, these credentials comprise both the usual username and password, and a second factor of authentication such as a PIN or a token. This (one-time) authentication element is normally used only when performing money transfers or other sensitive operations. As a security measure, many banking websites use separate forms, and do not ask for login credentials along with the second factor of authentication.

As of version 1.1.0, SpyEye incorporates the WebInject module, which can be used to manipulate and inject arbitrary content into the data transmitted between an HTTP(S) server and the browser. As described in detail in Section 3.4.1.2, a WebInject module is placed between the browser's rendering engine and the HTTP(S) API functions. For this reason, the trojan has access to the decrypted data, if any encryption is used (e.g., SSL).

In the case of information stealers, the WebInject module is leveraged to selectively insert the HTML or JavaScript code that is necessary to steal the target information. For example, as shown in Figure 3.12, the WebInject module inserts an additional input field in the main login form of an online banking website. The goal is to lure the victim such that he or she believes that the web page is legitimately asking for the second factor of authentication up front. In fact, the victim will notice no suspicious signs (e.g., invalid SSL certificate or different URL) because the page is modified "on the fly" right before display, directly on the local workstation.

WebInjects effectively allow attackers to modify only the portion of page they need by means of site-specific content-injection rules. More precisely, the attacker sets two hooks (`data_before` and `data_after`) that identify a portion of a web page where the HTML or JavaScript content (defined with the `data_inject` variable) is injected. These variables are set at configuration time into a proper file, named `webinjects.txt` in the case of ZeuS, SpyEye, and derivatives. Additionally, at runtime, the malware polls the botnet command-and-control (C&C) server for further configuration options—including new injection rules.

In contrast to phishing, which requires the attacker to host and maintain a spoofed web page, WebInjects do not require any external resource. Therefore, they reduce the upkeep effort for the attacker and also remove a point of failure (i.e., the external web page). Unfortunately, unlike phishing, which is indeed affected by take-down actions [91], the targeted organizations can do little to protect infected clients, because the injection itself is only visible on the client side.

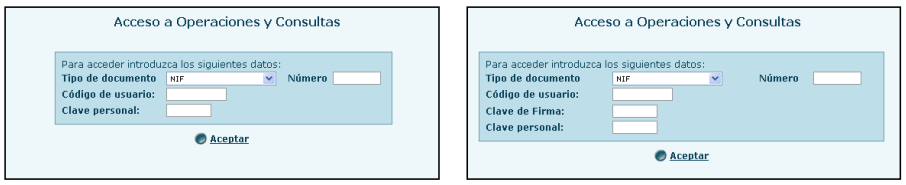
Because of their effectiveness and flexibility, WebInjects have gained a lot of popularity in the underground economy, and contributed to the malware-as-a-service phenomenon. To some extent, the configuration files embody the actual value of an information stealer. Indeed, these files, and in particular `webinjects.txt` files, are traded[2] or sold[3] on underground marketplaces.

3.4.1.2 Characterizing WebInjects: State of the Art and Challenges

From the malware analysis point of view, the threat landscape described above translates into an increased volume of distinct samples. In fact, not only the malware binaries can be packed and obfuscated with a wide array of options (e.g., packing method, encryption key), also the custom configuration files are encrypted, and embedded in the final executable. This characteristic, combined with the evolving nature of modern trojans, makes it very difficult to extract the static and dynamic configuration files—besides,

[2] http://trackingcybercrime.blogspot.it/2012/08/high-quality-webinject-for-banking-bot.html

[3] https://www.net-security.org/malware_news.php?id=2163




```
set_url https://extranet.banesto.es/npage/OtrosLogin/LoginIBanesto
data_before
name=usuario*</td>
data_end
data_inject
</tr><tr>
</TR> <TR> <TD align=left><FONT size=+0><B>
Clave de Firma:</B></FONT></TD> <TD align=left colSpan=3><INP
data_end
data_after
data_end
```

Figure 3.12: Example of a real WebInject found on a page of `extranet.banesto.es`, performed by a ZeuS variant (MD5 15a4947383bf5cd6d6481d2bad82d3b6), along with the respective `webinject.txt` configuration file.

of course, through custom, time-consuming reverse-engineering efforts, or in the lucky case that the malware itself exposes some vulnerabilities (e.g., SQL injection, weak cryptography). Examples of such approaches are proposed in [103]. They leverage a vulnerability of the encryption routines to create a chosen-plaintext attack. On the one hand, approaches that revolve around an initial, in-depth reverse engineering of a malware binary are useful to identify vulnerabilities or patterns of activities that can be exploited as detection criteria. On the other hand, the generality of these approaches is obviously limited.

[40] analyzed the WebInject functionality to fingerprint the behavior of information stealers. Their key intuition is that WebInjects are currently implemented by hooking into the Windows API functions. More precisely, since version 2, ZeuS hooks into the `Wininet.dll` library, which defines functions that are used by Microsoft Internet Explorer to handle web traffic (e.g., `HttpSendRequestA`, `InternetReadFile`), as summarized in Figure 3.13. The authors analyzed all the possible hooking mechanisms that could be implemented in the Windows OS (i.e., inline hooks, import address table hooks, export address table hooks, and hooking techniques that manipulate the windows loader mechanism) and, from them, they derived behavioral fingerprints. In practice, they look for extra code sections in the

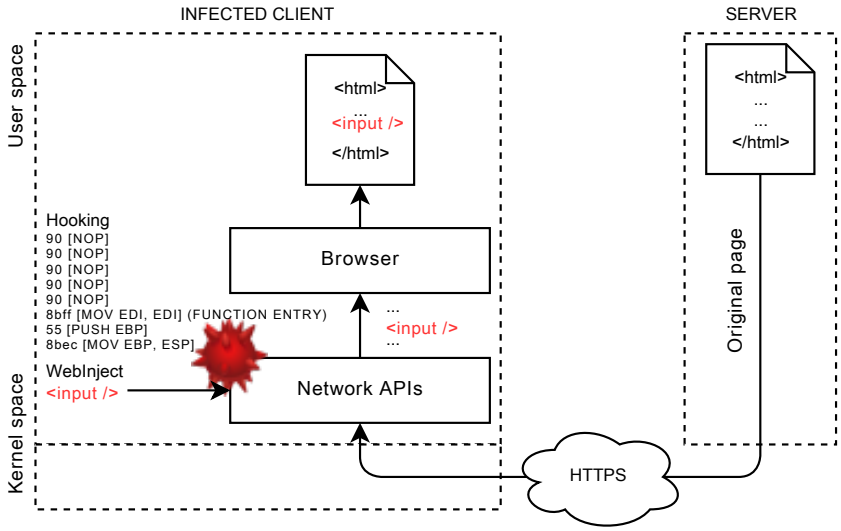


Figure 3.13: The HTML source code produced by the banking website transits encrypted over the Internet. When it reaches the OS and thus the `Wininet.dll` library, the source code is decrypted and intercepted. ZeuS modifies it on the fly and sends it through the same pipeline, up to the browser rendering engine.

basic Windows libraries, by comparing the version stored on disk against the version loaded in the process memory. Extra code sections are a sign of code injection due to hooking.

The generality of existing techniques depends on the fact that families of information-stealing trojans (as well as many other families of complex malware) hail portions of code from each other. For example, [40] notices that the implementation of the WebInject module of ZeuS is similar to that of SpyEye: They both rely on API hooking. Unfortunately, the knowledge that is necessary to build signatures of the action of the WebInject module can only come from reverse engineering of the samples. Although SpyEye and ZeuS feature a similar WebInject module, if an unknown sample or family implements a web-injection module differently, further time-consuming reverse engineering would be needed. Additionally, the whole approach is OS and browser dependent: Browsers other than Internet Explorer (e.g., Firefox is already targeted by the current versions of SpyEye, as described in [22]) use different libraries; OSs other than Windows have different userland APIs. Indeed, as highlighted in the latest ENISA Threat Landscape [86], the popularity of cross-platform malware increased in 2011–2012 (the most notable example is the Flashback botnet, which was reported that contained more than 600,000 Apple Macs).

Therefore, given the motivations presented in this section, we conclude that a generic approach for generating the signatures of an information stealer that performs WebInjects is needed.

3.4.2 Overview of Zarathustra

From hereinafter we use the term "WebInject" in its most general interpretation to refer to any mechanism used by malware to inject arbitrary content in the (decrypted) data that transits between the network layer and the rendering engine of a browser (see Figure 3.13). Although we focus on various ZeuS samples as a study case, our approach is not limited to this family: It applies to (possibly unknown) software that may implement an in-the-browser injection mechanism that results in additional code injected in a rendered web page.

The goal of our approach, called Zarathustra, is to generate signatures of the injection functionality automatically. A fast method to fingerprint the injection behavior of a malware is useful because it can tell whether an end host is infected by a known sample. This task is usually accomplished by antivirus software, whose signatures are manually constructed by analysts through tedious reverse engineering. In addition to producing signatures automatically, our system isolates precisely the injected code, as if the configuration files of the custom malware variant were available. This piece of information is particularly useful for security officers of bank institutions, because it allows them to verify automatically if and how their website is targeted by an information-stealing campaign.

The action of an injection module eventually results in changes to the document object model (DOM). This is the key observation behind our work. In particular, Zarathustra generates the aforementioned signatures by first rendering a website's page multiple times in an instrumented browser that runs on distinct, clean machines. In this phase, Zarathustra removes the legitimate DOM differences (e.g., due to ads, A/B testing, cookies, load balancing, anti-caching mechanisms). Zarathustra repeats the same procedure on an infected machine, and finally extracts and generalizes the resulting, malicious differences—which we call "fingerprints" (or signatures). The whole system runs on dedicated machines with no interactions with real clients.

The source code of Zarathustra is available at https://code.google.com/p/zarathustra/.

Zarathustra recognizes the behavior of any WebInject-based information stealer by looking for the visible effect of the WebInjects in the targeted websites, regardless of the underlying implementation of the injection mechanism (e.g., API hooking, DLL patching, other yet unknown techniques). Zarathustra does not leverage any malware-specific component or vulner-

ability to observe and characterize the injection behavior. Therefore, it is more general by design.

To remove the dependence on a specific (version of the) OS, browser, and malware, we lift the detection of WebInjects to a higher level in user-land. In particular, we position the observation point of Zarathustra on top of the process space of the browser. Specifically, by generalizing the example WebInject in Figure 3.12, we observe that the source code of a website rendered on an infected client differs from the source code of the very same page rendered on a clean machine.

Although this approach is in principle quite simple when applied at a small scale (e.g., by manual analysis of a handful of target websites and samples, as shown in an example in [94]), streamlining the generation of these fingerprints in the large scale presents two main issues:

- websites, by their own nature, may vary legitimately. This can be due to several factors, including server-side caching, web replication mechanisms adopted by the internet service providers or content-delivery networks and, mainly, upgrades of the (banking) web application. As noted in [85], these changes are very frequent in real-world web applications. The unfortunate side effect is that protection tools based on anomaly models yield false detections, because they confuse legitimate changes with tampered HTTP interactions.

- The rendering of a legitimate web page can vary depending on the inclusion of external resources performed on the client side (e.g., mashups), which is a common practice in websites nowadays (e.g., advertising, asynchronous content).

Both these issues yield differences that are not actual injections. This problem is hard to solve in general and its solution is beyond the scope of our research. However, in the limited scope of an attacker that needs to inject at least one piece of static code without disrupting the legitimate look of a web page, we can address the above issues as described in Section 3.4.3.2 and 3.4.3.3.

3.4.2.1 Workflow

As summarized in Figure 3.14, Zarathustra receives a trojan sample in the form of an executable file, and a target URL (list) as input. First, in the **DOM Collection** phase (Section 3.4.3.1), it collects a set of DOMs from a set of identical clean machines, and one DOM from the machine infected with the malicious executable. The DOMs are compared in the **DOM Comparison** phase (Section 3.4.3.2), which finds the differences between the "clean DOMs" and the "malicious DOM". In the **Fingerprint Generation** phase (Section 3.4.3.3), the differences are analyzed to eliminate obvious

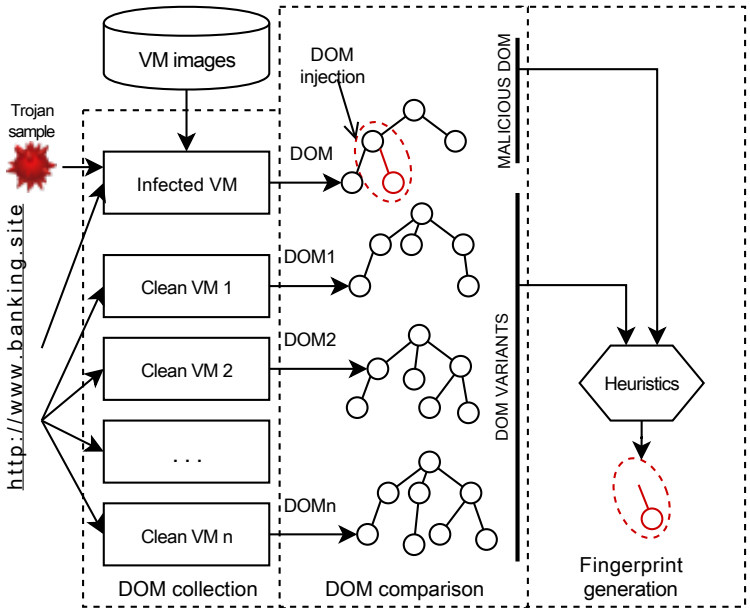


Figure 3.14: Server side architecture of Zarathustra, which is in charge of analyzing a given URL against a given trojan.

duplicates (e.g., due to legitimate changes or caching) and other recurring patterns of legitimate differences.

3.4.3 Implementation Details

We implemented the **DOM Collection** phase on top of Oracle VirtualBox. We wrote a thin library on top of its API to create, snapshot, start-stop the VMs, and a library based on WebDriver[4], a platform- and language-neutral interface that introspects into, and controls the behavior of, a web browser and dumps the DOM once a page is fully loaded. The **DOM Comparison** relies on XMLUnit's `DetailedDiff` class functions. The **Fingerprint Generation** phase does not rely on 3rd-party libraries.

3.4.3.1 DOM Collection

This phase receives a target URL as input. It starts n clean VMs plus one infected VM. Each VM automatically starts the browser with WebDriver, sleeps for T seconds, visits the URL, lets the page load completely, and saves the resulting DOM. We then access and store the DOM as computed by the browser, thus including all the manipulations performed by client-side code

[4] http://www.w3.org/TR/webdriver/

at runtime while the page loads. The DOM encompasses the content of the nodes in the page, including script tags. This phase outputs the n "clean DOMs" that result from visiting the target URL with the clean VMs, plus one "malicious" DOM for the infected machine.

As [40] measured that ZeuS typically waits 30–110 seconds before hooking the OS functions, allowing more time after starting the browser may increase chances to detect injections. However, we manually verified that, in our dataset, the injections were visible even after $T = 3$ seconds. So, we left this timeout as a parameter, which can be increased in production environments.

3.4.3.2 DOM Comparison

This phase compares DOM, the "malicious DOM" against the "clean" $DOM_i \in [1, n]$ to find distinct differences. We rely on XMLUnit's `DetailedDiff.getAllDifferences()`, which walks the tree of DOM and, for each node, walks the tree of DOM_i to look for the following differences:

New node: This is extremely important to catch one of the most common manifestations of an information stealer: new `<input />` fields. This phase takes into account any element type (e.g., forms, scripts, iframes, text).

New attribute: This type of difference reveals the presence of possibly malicious attributes such as the `onclick` trigger, used to bind JavaScript code that peforms (malicious) actions whenever certain user-interface events occur.

Attribute value modification: This type of difference occurs when the information stealer manipulates an existing attribute (e.g., to change the server that receives the data submitted with a form, or modifies the JavaScript code already associated to an action).

Text node content modification: This occurs when a malware modifies the content of an existing node, for instance to add new code within a `<script />` tag.

We did not observe removal of DOM nodes in our dataset, nor there is any malware capability in this direction as the WebInject configuration file only allows to add nodes. Indeed, DOM node removal not followed by any node addition is against the goals of the malware operator. A DOM node substitution is accounted for by Zarathustra as a modification (3rd and 4th case).

The output of this phase is a set of differences $\mathrm{diff}(DOM_i, DOM), \forall i \in [1, n]$, where "$\mathrm{diff}(\mathrm{A}, \mathrm{B})$" indicates the distinct differences between DOM A and B.

3.4.3.3 Fingerprint Generation

This phase processes the differences from the **DOM Comparison** and generates a set of fingerprints $F = \cup_{i=1}^{n} \text{diff}(DOM_i, DOM)$. Before computing the union, we remove the differences between each couple DOM_i and DOM_j, $\forall i \neq j$ to take into account the legitimate changes between "clean DOMs", which could cause false (positive) signatures. An example generated fingerprint is

```
"clean_node": {
  "parent": "null",
  "value": "null",
  "xpath": "null"
},
"malicious_node": {
  "parent": "form",
  "value": "input",
  "xpath": "/html[1]/body[1]/center[3]/table[1]/tbody[1]/tr[1]/form[1]/input[13]
}
```

which specifies the `<input />` field injected in the real case of Figure 3.12. The set of fingerprints F already contains valuable information that precisely characterizes if and how an injection takes place.

As F is generated in a fully automated way, it may contain false differences. These are addressed by two heuristics.

Heuristic 1: Ignoring Dynamic DOM Differences. Our system can disable the JavaScript interpreter during fingerprint generation. We observed that several legitimate differences found in our dataset are actually due to DOM modifications performed by the browser that executes JavaScript routines while rendering the page. At a first glance, disabling JavaScript may lead to excluding malicious DOM modifications caused by the malware. However, a genuine WebInject always results in at least one static code injection, which would be still visible even when JavaScript is disabled. As we discuss in Section 3.4.6, even in the corner case of a malware that injects code inside an existing `<script />` tag, by adding static code that performs the actual DOM manipulation, Zarathustra still generates a signature of the static code injection in the first place.

Heuristic 2: Caching Server Responses. From our experiments, we observe that the **DOM Comparison** phase needs at least $n = 30$ clean VMs in order to correctly distinguish between legitimate and malicious differences. However, visiting or rendering the same site n times may not be feasible (e.g., banning, hardware restrictions). By caching server responses—using the URL as the caching key—we reduce the false differences due to dynamic code on the server side, which may insert, for instance, a unique identifier in each response (e.g., to avoid cross-site request forgery or caching).

3.4.4 Post-processing Heuristics

The output of the previous phase is the set F of fingerprints, which is post processed through the following two heuristics that minimize the occurrence of false differences.

Heuristic 3: Filtering Special Attributes. Several attributes can be safely ignored, because they would not lead to new DOM nodes. We assume that if a malware attempts to forcefully inject a DOM node (e.g., `<input />`) into an attribute value, this would lead to parsing errors, and thus to a useless DOM node. Specifically, we ignore `value`, `style`, `class`, `width`, `height`, `sizset`, `sizcache`, and `alt`. The `style` attribute may be used maliciously, to inject JavaScript code. However, **Heuristic 1** prevents this case.

Heuristic 4: Filtering Text Nodes. Text nodes are harmless, because they can only contain pure text. We ignore all the text nodes, unless they are children of `<script />` tags. There are many other ways through which a malware can insert custom client-side code, but Zarathustra already accounts for these types of WebInjects during **DOM Comparison**.

3.4.5 Experimental Evaluation

Between January and February 2013 we evaluated our implementation of Zarathustra against 213 real, live URLs of banking websites and 56 distinct samples of ZeuS (see Table 3.4). Our main goal was to measure the correctness of the fingerprints, with and without the heuristics described in Section 3.4.3. We also wanted to assess the resource requirements of Zarathustra in order to analyze a given amount of URLs and samples.

3.4.5.1 Dataset

With the above premises, our decision fell on ZeuS, because it is by far the most widespread information stealer that performs injections: According to ZeuS Tracker, as of September 23, 2013 there are 887 known C&C servers (449 active), and an alarmingly low estimated antivirus detection rate (38.29%, zero for the most popular and recent samples). We also conducted a series of explorative experiments with SpyEye, which is less monitored than ZeuS (216 C&C servers, 77 active, and an average detection rate of 27.1%); thus, it is more difficult to obtain an ample set of recent samples. However, SpyEye features the same WebInject module of ZeuS, as described in [114, 35, 40]. For these reasons, for the purpose of evaluating the quality of our fingerprint-generation approach, we decided on ZeuS as the most representative information stealer that generated real-world injections.

We constructed the list of the target URLs by merging 2 `webinjects.txt` files found on underground forums, plus the `webinjects.txt` leaked as part

MD5	Injections	MD5	Injections
68ab93087e2bf697e48b912b4546e666.exe	0	4df1446e8419978a0999ff2fa3fd60a3.exe	17
93895e081e679f8d9760de48b4ad349f.exe	17	041c17a7b97550fd69d25613d9ef8f46.exe	0
757f4dcb8fb34e8d168e632f16cebd53.exe	13	9bc0e3d19af915c608a784fda63b0076.exe	13
1a45e46567b84d38ba868f702e795913.exe	4	a4aa162745adcb84373e6a623125c650.exe	12
fd622057a281813c32cade7ad54843a5.exe	12	22788996e2381bdb97480b8de141ec2c.exe	0
9cd8fbd475c088d860bdc1371924dd4f.exe	13	5e26d372feb7d085b752fffa931fc156.exe	0
9ffe865c925bf06d35aa6b68cdaf3609.exe	0	39ad78a889a2b40a94dd700d67f1a5ed.exe	2
85719c933ccdb42f37e8c4d9b5e6bcfd.exe	0	b2c82ffe10763cdc241c7fa8d97097ae.exe	13
2105082b794ecfa02136e012f5ab4e6b.exe	0	bf45f27a403acfd3847fbbae88a8375f.exe	0
15a4947383bf5cd6d6481d2bad82d3b6.exe	13	9abaffda80841aa87c9f5786e0db639e.exe	0
b2a52dabdc8134199cd7858dd8e41013.exe	17	029d4f8dcf43837f773116439b07e980.exe	1
b68d88be4d65b29ad17937d8a419d8ba.exe	0	08e01221186cf82952c25d995176561b.exe	0
bb0c5a0c13682b996f5ab4b5dd79f430.exe	17	6436032a3d5bf53c6273ddd0ffab80be.exe	40
254712088ab8e08619f20705d7a09cf1.exe	0	fd12f0d2e2bbef953ac87d4dca32c15d.exe	0
6ba342b445092151d8171a62efe633cb.exe	17	3ba3149213e6b9091c727104dbb26ea6.exe	41
71d1a97b5776f3adc7f92ba6e82d162b.exe	13	b62dbd301f130487dfbc1473dced8aad.exe	17
b82eeaf8d5c0ed3d44269196865beb80.exe	13	f75e3fa05762072e5e6471f3fb982087.exe	13
21ef35e6e3f3494d134e9928ca6f38e8.exe	17	c04fddfaab6b879a25b036980a34908e.exe	12
e54d1b119211907dad7dc33ff087d5be.exe	13	ffcaf8a2f2f59e0f7b165d085842bd17.exe	16
56f8a7c7721aa96e543d57b0fef0f98f.exe	0	70dfde201f6a9a66730d9ae6b69450f8.exe	42
2a12ba5847c0fb58a89ea6b2f6dd1a97.exe	0	ecc0a5bdf5174efcd9d292e815de06f4.exe	11
1ad8e54179e8c2c7767ea3b039d542fc.exe	2	5298f1fd6b300223f6bcdbc1fa89c2c0.exe	0
9b9951c50e04818c413c8cd1a3096a6b.exe	0	7f280b73093e5b61ab2eec7b6ebda420.exe	17
d60487f05000160d85db0b354dbdd866.exe	16	21248f3752c84ee5866a95992dba0813.exe	17
cdf3bb9c75000fc49c7c148b76c20b45.exe	17	51eef801f614a0278c8b79f7be9d2fdf.exe	12
31ea03a2a33a75ddf48d52f4605ef0bb.exe	16	be4f416d394b4e305fd0e11d40a4242c.exe	17
b1a49aa03fc1a8226ebc1205bdcf5562.exe	13	99646549006435d73efeddbbbcf4313f.exe	13
6384e4f1b5eeefbcb99a281ac514078a.exe	0	c4ba4d84e5b40132e82b403469eb13ca.exe	0

Table 3.4: Evaluation dataset overview. All of the samples are variants of the ZeuS family.

of ZeuS 2.0.8.9 source code[5]. We so obtained 293 distinct URLs. To make it feasible to manually verify the results in a reasonable time, we selected 213 URLs (143 organizations) among the URLs that were active at the time of evaluation. Building a list of URLs from `webinjects.txt` files found in the wild allowed us to deal with real-world targeted pages. As WebInjects occur on landing or login pages, we concentrated our search on such pages.

We created the ground truth by configuring Zarathustra with all the heuristics enabled. We then manually analyzed the results to ensure that no false signatures or negatives were found. An alternative approach could have been to craft a proper `webinjects.txt` as the ground truth. However, we wanted to test Zarathustra on injections found in the wild.

3.4.5.2 Setup and Scalability

We deployed Zarathustra on a 1.6GHz, 4-core x86-64 Intel machine with 16GB of RAM. We installed Windows XP SP3 (Internet Explorer 6, 7, 8) on each VM and granted outbound and inbound Internet access. Zarathustra required 256MB of RAM and 2 to 5GBs of disk space per VM; in our experiments we used 2 to 35 VMs. We downloaded 76 samples, but 20 of these failed to install or crashed, leaving 56 distinct samples. Figure 3.15 shows that Zarathustra scales well: With 10 VMs running in parallel we are able to process 1 URL in less than 3 seconds. The architecture of Zarathustra has no

[5] https://bitbucket.org/davaeron/zeus/

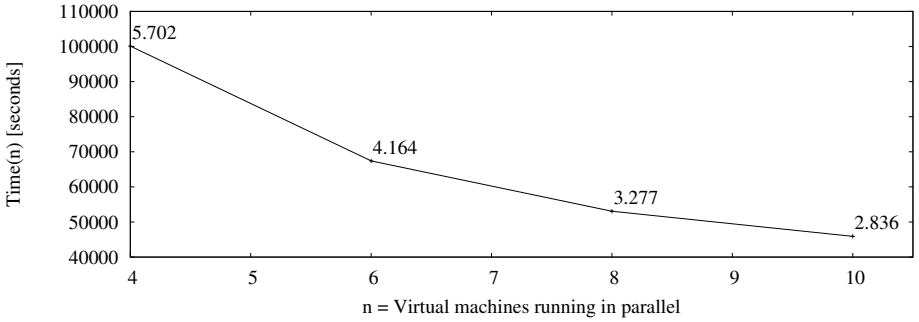


Figure 3.15: Scalability of Zarathustra: Time required to process 213 URLs with 76 samples (including crashing samples). the labeled points indicate the time to process 1 URL.

central node, nor any dependency that prevent full parallel operation: As a result, its capacity scales directly with the amount of resources available.

3.4.5.3 Correctness of the Signatures

Table 3.5 summarizes the top-ten domains where Zarathustra correctly recognized injections caused by ZeuS. Some samples perform zero injections, although usually we found around 1 to 9 injections per distinct URL of the same domain.

Table 3.6 summarizes the influence of each heuristic: We disabled one heuristic at a time and ran the same experiment. The last row reports the correctness of the signatures when all the heuristics are enabled: We manually verified the presence of actual injections and set this as the ground truth for the experiments reported in the above rows. Overall, Zarathustra correctly detected that ZeuS was performing an injection in 23.48% of the URLs. The second column is the most important one. It shows the fraction of URLs where Zarathustra correctly detected that a specific sample was performing an injection. We notice that the contribution of the first heuristic is fundamental, because such fraction of URLs decreases to 39.58% (on average) when disabled. The second heuristic also provides a significant contribution, whereas the last two heuristics are not particularly influential in our dataset.

3.4.5.4 Signatures that Cause False Positives

A false positive occurs mainly when Zarathustra generates a signature of a legitimate, benign difference. Obviously, the false positives have lower impact when raised on an infected machine than on a clean machine. The rationale is that an alarm usually leads to some reaction (e.g., investigation, disinfection), which would waste time and resources on a clean machine.

EFFECTIVE TLD	INJECTIONS			
	min	max	tot	avg
ybonline.co.uk	0	28	952	9.0667
cbonline.co.uk	0	45	699	2.6885
lloydstsb.com	0	23	677	4.3121
bbvanetoffice.com	0	14	312	5.7778
virginmoney.com	0	279	279	5.6939
if.com	0	77	231	4.2778
banesto.es	0	10	194	0.7239
rbkmoney.ru	0	8	112	2.1132
accessmycardonline.com	0	31	93	1.7547
smile.co.uk	0	29	87	1.6415

Table 3.5: Top ten websites in our dataset. The no. of injections is calculated and averaged over the set of ZeuS 56 samples, and on the URLs within each domain.

HEURISTICS	AVG. CORRECT ($\pm$ VAR.)	%URLs
2,3,4	**39.58 $\pm$ 11.53%**	52.17%
1,3,4	74.98 $\pm$ 15.42%	23.48%
1,2,4	97.97 $\pm$ 0.069%	22.61%
1,2,3	98.42 $\pm$ 0.124%	23.04%
All	100.0%	23.48%

Table 3.6: Contribution of each heuristic on the quality of the signatures. The second column reports the fraction of URLs with correctly-identified injections (this fraction is averaged over the set of 56 samples). The last column reports the fraction of URLs where at least one sample was detected while performing an injection, including false signatures, which are analyzed separately in Section 3.4.5.4.

On the other hand, a false alarm raised on the wrong site is beneficial anyways if it occurs on an infected machine. In both cases, on the data collected during the experiment described in Section 3.4.5.3, we obtained zero false positives when using all the heuristics on the entire dataset.

In a more detailed analysis, we concentrated on the influence of **Heuristic 1**, which was the most effective at eliminating false positives, as the first row of Table 3.6 shows. For this, we disabled **Heuristic 1**; then, on all the URLs in our dataset, generated the signatures using an increasing number $n \in [1, 35]$ of clean machines. In this way, we can assess how well Zarathustra can tell legitimate differences and true positives apart when using a sufficiently large number of emulated clean clients. We then prepared four machines, each infected with a distinct ZeuS sample, to evaluate the false positives also on infected machines.

As Figure 3.16 shows, both the false positives decrease while n increase. More importantly, the false positive rate on clean machines drops to almost zero (1%) if at least 35 clean machines are used to generate the signatures. We manually observed that the vast majority of that 1%, at $n = 35$, was caused by JavaScript-based advertisement networks and modifications performed by the browser, which lead to highly-dynamic DOMs. Thus, when deploying Zarathustra to protect from injections on web pages that have a dynamically-generated DOM, it is recommended that either **Heuristic 1** is enabled, or a large number of machines is used.

Overall, we can conclude that the most relevant class of false positives is low. Signatures extracted by state-of-the-art approaches (e.g., via reverse engineering) may have lower false positives. However, considering the time required to generate signatures with these methods, the price is that of missed signatures (i.e., false negatives). Zarathustra, instead, builds signatures of known WebInject-based malware automatically and quickly.

3.4.6 Discussion and Limitations

The first critical issue with Zarathustra is that malware operators could rewrite the injected code, introducing no-op DOM nodes with the goal of evading the signatures generated by Zarathustra: Adding an additional `<div />` wrapper to a page (in a random position), for instance, would circumvent a naïve use of our signatures. However, none of the samples in our dataset adopted this technique. In addition, and more importantly, modifying the structure of a page can easily result in user-visible, brittle modifications. This is clearly against the malware operator's goal of preserving the look of the page as much as possible. Although we leave the implementation of a proper signature matching algorithm to future work, we are aware that there is an accuracy trade off between matching the entire XPath expression of a signature versus matching only the leaf nodes. However, if the leaf

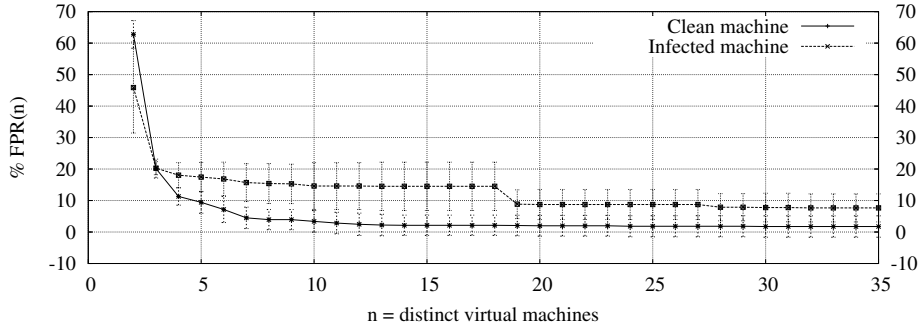


Figure 3.16: False positives due to legitimate differences decrease for an increasing number, $n \in [2, 35]$, of clean VMs, until it reaches 1.0%. We used 206 distinct URLs, rendered on a machine infected with ZeuS (MD5 a4aa162745adcb84373e6a623125c650). With **Heuristic 1** enabled, we achieve zero false positives.

nodes are detailed enough (e.g., they contain attributes), an algorithm that matches on the leaf nodes can achieve good accuracy and high generality.

The second issue is that Zarathustra focuses on generating signatures of information stealers automatically, but it is still a reactive approach: given a sample of the malware, we can generate a signature, but there is no guarantee that this signature will also proactively match future generations of the sample. However, the scalability of Zarathustra allows analysts to automatically determine the targets of a large set of information stealers, with no reverse engineering required.

Another obstacle that Zarathustra has to face are evasion mechanisms employed by the malware to fool dynamic analysis. For example, a sample may refuse to expose its true malicious behavior when it detects the action of well-known debugging tools or analysis environments. However, we do not rely on debugging or introspection tools: We rely on virtual machines solely for ease of implementation and flexibility during evaluation. Zarathustra works perfectly, and even faster, on bare metal. Hence, this obstacle is easily circumvented by adopting the method proposed in [76] to obtain a virtual-machine-equivalent snapshots on physical hardware. This way, no malware can possibly recognize that it is running in a controlled environment.

WebInjects are the only artifacts that we rely on to observe the action of an information stealer. As a result, if a banking trojan succeeds in hiding its behavior (e.g., by injecting content only in some requests), Zarathustra cannot guarantee to extract differences every time a targeted URL is visited. However, this limitation does not constrain our system only; it also affects state-of-the-art mechanisms based on APIs hooking detection. Hooking of APIs may not always take place, or be delayed as noticed in [40]. In this

regard, Zarathustra provides the same result of other techniques, while requiring less implementation efforts and more portability.

Last, a minor point of our current implementation of Zarathustra is that we use the (banking) website as an oracle. For reasons that fall outside our attacker model (e.g., client-side malware), an injection may match exactly with a benign difference. For example, this happens if the website is updated with a new form input that has the very same DOM representation of an injection. Not only is this very unlikely to happen, it is also very easy to remediate, by leveraging feedback from the bank whenever their site is updated, or possibly by requesting an update of the signatures for that domain. It is indeed reasonable to imagine Zarathustra being deployed within a bank information system: this use case would erase most, if not all the possibilities for false positives as a fully up-to-date model of the clean website would always be available. In a similar way, Zarathustra can easily monitor authenticated web pages, which are not a limitation when our system is deployed by the website provider (e.g., bank).

3.4.7 Related Work

This section complements Section 3.4.1, where we mention the work that is most relevant to ours.

With particular attention to ZeuS and SpyEye, trojans have been studied in the past two years. [114] gives a detailed overview of the components of SpyEye, including its development kit, and describe how SpyEye integrates in the whole criminal ecosystem. [35] performed a similar study on the ZeuS crimeware toolkit.

In Section 3.4.1 we already introduced BankSafe [40], a recent approach that specifically targets the mitigation of information stealers, which inspired on to create Zarathustra. BankSafe is similar in spirit to Zarathustra: It also looks for observable changes, although BankSafe focuses on the binary libraries loaded in memory. The assumption is that information stealers leave traces in loaded libraries by means of hooked functions. Because of this assumption, BankSealer is limited to the Windows OS and, more importantly, is highly dependent on the hooking mechanism and functions adopted by a specific malware binary. The approach proposed in [67] protects the browser from malicious websites that perform dynamic changes to the DOM. Although not designed specifically to target information stealers, it could be applied to recognize WebInjects. The system instruments the ECMA script layer by proxying its functions so to profile their execution and recognize malicious patterns. However, the authors mention that their method can detect dynamic changes of the DOM, whereas WebInjects work at the source-code level.

Along a different line, [103] developed a chosen-plaintext attack against the encrypted stream that flows between ZeuS (1.x and 2.x) and its C&C.

The chosen plaintext is a combination of the information from the analysis of the malware toolkit and the data collected while running a sample in a controlled environment (e.g., cookies, user credentials, or computer hostname). As discussed in Section 3.4.1.2, these attacks are effective, but their applicability requires the reverse engineering of the malware and parts of its ecosystem in order to retrieve sufficient knowledge.

Malware analysis is a broad research area that was recently systematized in [107], who surveyed and discussed 36 academic publications presented at the 6 highest-ranked security conferences between 2006 and 2011. The goal of the work was to set the guidelines for performing rigorous malware experiments. We strive to adhere to such guidelines while evaluating Zarathustra, as described in Section 3.4.5.

3.4.8 Future Work

Besides addressing the limitations described in Section 3.4.6, there are two future directions that we deem promising.

First, the most practical future work that needs to be evaluated is a system that protects the clients of a website from information stealers in a fully centralized manner, with no additional software component installed on the client side. In our vision, such a system would consist of a reverse proxy that "appends" a JavaScript routine before sending HTTP responses to the client. This routine would (embed and) match the known signatures for the current URL once the browser has rendered the requested page. This scheme poses two challenges: first, the matching must be computationally inexpensive and fast; secondly, the insertion of the JavaScript code must be performed in such a way that its removal would require major re-engineering of the malware internals, or at least trigger some integrity check.

Second, as described in [75], some attackers are shifting towards more advanced WebInject methods, operating more subtle changes which do not result in user-visible DOM modifications. Although some of these methods can be directly detected using the approach described in this section, others result in advanced manipulation of the HTTP requests to divert monetary transactions to a bank account under the attacker's control. The respective HTTP responses (e.g., page that confirms the result of a transaction) and all the subsequent interactions with the banking website are also modified such that the true recipient of malicious wire transfers is masqueraded (i.e., replaced with the intended recipient's name). This threat will require modifications to Zarathustra, because the injections may occur in pure text nodes. Thus, the set of heuristics will need to be refined to cope with these corner cases.

Finally, in Zarathustra we showed that the DOM is a simple yet effective observation point. However, we believe that other aspects of the browser behavior can be observed and compared on infected vs. clean clients, to

assess whether the information stealers cause side effects in the browser that can be used as a detection criteria.

4

The Role of Phone Numbers in Understanding Cyber-Crime Schemes

Both Internet and telephones are part of everyone's modern life. Unfortunately, several criminal activities also rely on these technologies to reach their victims. While the use and importance of the Internet has been largely studied, previous work overlooked the role that phone numbers can play in understanding online threats.

In this chapter we aim at determining if leveraging phone numbers analysis can improve our understanding of the underground markets, illegal computer activities, or cyber-crime in general. This knowledge could then be adopted by several defensive mechanisms, including blacklists or advanced spam heuristics. Our results show that, in scam activities, phone numbers remain often more stable over time than email addresses. Using a combination of graph analysis and geographical Home Location Register (HLR) lookups, we identify recurrent cyber-criminal business models and link together scam communities that spread over different countries.

4.1 Problem overview and state of the art

In many fraud schemes phone numbers play an important role. For example, criminals have been analyzed by authorities based on their phone numbers on public or underground forums [17]. In other online fraud cases, like one-click fraud [44], usage of a phone number can make the fraud appear more legitimate to a victim. Finally, scammers will often use the phone to defraud victims [115].

However, traditional *a-lá-Mitnick* scams are based on pure social engineering techniques and, despite their effectiveness, they are relatively slow. To make this a viable business, modern scammers have begun to take advantage of the customers' familiarity with "new technologies" such as Internet-

based telephony, text-messages [68], and automated telephone services. Another example is the use of instant messaging (e.g., Windows Live Messenger, Skype, the FaceBook chat), which involves some form of conversation with computer programs that leverages natural language processing and artificial intelligence techniques to mimic a real person [81].

A particular variant of phishing, known as *vishing* (i.e., voice phishing), was popular in the U.S. in 2006–2009 [71], and is now slowly gaining ground in Europe. Notably, an experiment conducted in 2010 by the United Nations Interregional Crime and Justice Research Institute revealed that the 25.9% of Italians (on a sample comprising 800 randomly-selected citizens) were successfully tricked by phone scammers.

A recent study of fraud activity in Japan [44] demonstrates that phone numbers can play an important role in online fraud and can be used as a way to link and identify criminals. While there are several indications of criminals using phone numbers for their malicious activities [17], we still lack a global understanding to compare the usage and the role of the phone numbers in different criminal schemes. Previous work is limited to the study of spam over SMS, or to phone number abuses through premium services [110] [98] [70].

To address these growing concerns, we follow two lines of research.

In the first, we use as a case study the well known Nigerian scam attacks. First, we want to evaluate the reliability of leveraging automated phone numbers analysis to improve our understanding of the underground markets, illegal computer activities and cyber-criminals in general. Second, we aim at finding patterns associated with recurrent criminal activities, in particular we automatically identify the communities responsible for Nigerian scam campaigns. Finally, we correlate the extracted information and enrich it with geographical and phone number life-cycle information from HLR lookups, to validate our hypothesis of phone numbers being actively re-used instead of discarded. Along these three directions, we summarize our main findings and contributions as follows:

- We present a study of the use of phone numbers on Nigerian scam attacks.
- We show that phone numbers are a good way to automatically detect communities of scammers and study their behavior.
- To the best of our knowledge, we are the first to propose and use HLR lookups to verify our findings, and to study the use of phones and phone numbers over time by different and distributed criminal groups.

As a second line of research, we are developing a data collection system to capture different aspects of phishing campaigns, with a particular

focus on the emerging use of the voice channel. The general approach is to record inbound calls received on decoy phone lines, place outbound calls to the same caller identifiers (when available) and also to telephone numbers obtained from different sources. Specifically, our system analyzes instant messages (e.g., automated social engineering attempts) and suspicious emails (e.g., spam, phishing), and extracts telephone numbers, URLs and popular words from the content. In addition, users can voluntarily submit voice phishing (vishing) attempts through a public website. Extracted telephone numbers, URLs and popular words will be correlated to recognize campaigns by means of cross-channel relationships between messages.

4.2 Lessons learned from analyzing the Nigerian scam

4.2.1 Phone Numbers: Extraction and Quality

Phone numbers are often used, both directly and indirectly, in many cyber-criminal activities. For example, they appear in the registration of malicious domains, in the signatures of spam messages, in malware for mobile devices, and as main contact in scam and phishing campaigns. In some cases they are provided just to increase the credibility of some fake information, while in other scenarios they may represent a core component of the malicious activity itself.

At the beginning of our study we collected data from several sources related to illegal online activities. In particular, we focused on scam messages, spam messages, registration information of malicious domains (WHOIS) and Android malware. We selected those data sources because they are very likely to contain phone numbers and they are strictly related to cyber-crimes or fraud schemes.

After a first screening of the data, we observed a great variability in the quality and reliability of the collected information. To better describe this phenomenon, we classified the phone numbers along two directions: how difficult it is to extract them from raw data, and how reliable they are once they are properly extracted.

4.2.1.1 Extracting Phone Numbers

Properly recognizing and extracting numbers from a raw data stream proved to be quite challenging, which is consistent with results in [102]. The results mainly depend on three orthogonal factors:

How structured and easy to parse the information is For example, WHOIS records are very easy to process and the phone number is always located inside a known and well defined field. At the other end of the spectrum, phone

numbers stored in malicious binaries can be obfuscated and are, in general, very difficult to extract automatically.

How well formatted the number is A simple regular expression can be used to extract a fully qualified number with a clearly separated international prefix (e.g., "+1 (805) 403-1234"). Unfortunately, numbers can be written in many different forms, which can be combined thus making automated parsing even harder. Phone numbers can include international prefix '+' or '00' codes, only local prefix codes, or only the phone number digits. After that, phone numbers can be grouped in variable-length groups of 2, 3 or 4 digits. Additionally, the prefixes and groups can be separated by spaces, '.', '-' or other delimiting characters, which can be country specific as well. A number without its international prefix may potentially correspond to many different numbers in different countries. Therefore, a normalization algorithm has to be used to transform the extracted number into a non ambiguous fully qualified E.164 number. When adding a country code to a candidate phone number, a *numbering plan* can be used to check if the resulting number is a valid number or not (e.g., the range is allocated and it has the correct number of digits). Unfortunately, repeating this step with too many possible country codes leads to many false positives. This is a common problem in localized cyber-crime (e.g., malicious mobile application targeting the Chinese market) because the lack of an international prefix may force the analyst to try many possibilities, thus decreasing the reliability of the collected information. Finally, short numbers (e.g., 57341) can be very challenging to detect. In fact, since the length and format are country-specific, these numbers can be easily confused with other short sequences of digits.

How noisy the data source is This is a measure of how often the source data includes strings of digits that can be misinterpreted as phone numbers, such as identification or reference numbers, and IP addresses. This is often a problem when parsing email messages that contain several numbers mixed with text. The presence of many sequences that may resemble valid phone numbers can greatly increase the number of false positives of the automated extraction routine.

A number of heuristics can be used to improve the extraction process. For example, the immediate context of a phone number can be very useful to detect the presence of a phone number. Such context may include abbreviations or words to indicate a phone number is following (e.g., *phone, mobile, tel, fax, mobile, call, contact, line, dial, direct, ext*), combined with punctuation marks (e.g., *'.', ':'*).

The language used in the text surrounding the extracted number can also be used as a good indication of the geographic areas in which the number

is supposed to be used. This is especially true for phone numbers used in scam activities, when the scammer expects the victim to call that number without ambiguity. For example, for a message written in Russian language, that includes a phone number without a full international prefix, one can try to complete the number by considering those countries where the Russian language is widely spoke, e.g., Russia '+7', Ukraine '+380', Belarus '+375', Moldova '+373'.

However, there is always a trade-off between the amount of extracted numbers and the accuracy of the results. Even by applying properly tuned heuristics, the amount of false positives when extracting poorly formatted numbers from noisy sources can be very high.

4.2.1.2 Phone Number Extraction Reliability

After a set of candidate numbers are extracted from the raw data, it is important to distinguish the real numbers from the fake ones. This is largely dependent on the type of activity and on the reason why the phone number was used by the attacker.

For example, numbers present in spam messages can be randomly-generated or spoofed to mimic existing phone numbers and to deceive anti-spam filters. Also, when registering a domain name there is often no validation of the authenticity of the provided numbers. However, in certain forms of cyber-crime the number has to be real and somehow controlled by the attacker. This is the case of premium numbers used in mobile malware or contact numbers used in scam campaigns.

Since distinguishing a fake or spoofed number from a real one is very hard, we decided to focus our analysis on a data source containing more reliable numbers. Unfortunately, the mobile malware dataset is very small and most of its data consists of short numbers. Therefore, in the rest of the chapter we adopt the SCAM dataset for our study.

A potential improvement to reliable extraction could be achieved via *dynamic analysis validation*, i.e. calling the numbers. However, this technique is not feasible for many reasons, ranging from illegality of unsolicited calling or wardialing to financial infeasibility to call so many numbers. It is left as a separate future work.

4.2.2 Data Enrichment

The SCAM dataset consists of data from user reports. There are several *user reports aggregators* that cover a wide range of fraudulent activities. This information is usually reported in dedicated forums, blogs, and other online media sites. We selected the community-supported site `419scam.org` because it has a large dataset of well formatted scam reports. This dataset was manually collected, filtered and pre-processed from January 2009 to

August 2012. The dataset includes meta-data on each entry, i.e., the category, message headers and, for 16% of them, the corresponding original email body.

The original dataset was enriched with the service type (e.g., mobile, land line, premium) of each phone number using two different databases (so called *numbering plans* or *NNPC*). The first one is a free and open source XML-based database included in *libphonenumber* which derives the service type during the extraction and normalization process. The second one, is a commercial database [21] which is more complete. We use both sources to cross-check the results and detect possible discrepancies.

In our SCAM dataset, we identified in total 67,244 unique normalized phone numbers. Out of them 34,424 were UK PRS (*Premium Rate Services*) numbers (51% of total) and the rest 32,820 were non UK PRS numbers (49% of total). Out of the 32,820 non UK PRS numbers, there were 29,685 mobile phone numbers.

Finally, we collected additional information about the mobile numbers by performing an HLR lookup. HLRs are databases maintained by mobile operators containing information about the current status of a phone number – i.e., the International Mobile Subscriber Identity (IMSI), roaming status, and roaming operator. This can be very useful for our study, because this allows to know if a mobile phone number is still active and if it is roaming to a foreign country. However, HLRs are only accessible from within the SS7 telecommunication network, and therefore we had to rely on a third party commercial service [2] to query this information.

A detailed description of how HLR lookups are performed can be found in [14]. The basic idea is to contact the homing operator of a phone number pretending to be interested in initiating either an SMS or a voice call (e.g., by sending a `MAP_SEND_ROUTING_INFORMATION` message). At this point, the homing operator of the subscriber number checks the status of the mobile number and returns the details.

By performing an HLR lookup periodically for a given mobile phone number, we can get insight on the evolution of it's network status. Such status information can be used to draw conclusions about activities related to a mobile phone number. We describe the use and results of this technique in Section 4.2.5.

4.2.3 Fraud business models

In this section we summarize some of the fraud business models we observed in this work. Such models were identified using information from various sources (e.g., forums, and abused users complaints) as well as the observations we made while analyzing our datasets. While some of those business models are known, many were not well identified or were lacking empirical evidence.

4.2.3.1 Premium Phone Numbers

Premium phone numbers can be categorized as follows:

National Short Premium numbers can provide high profit but are difficult to set up. However, some third party businesses offer simple point-and-click interfaces to register and configure such services.

National Premium numbers can provide moderate to high profit, with low operational costs, and quick set up.

International Premium numbers are complex to set up and have high operational costs. Moreover, they are blocked by some telecom operators.

UK Personal Numbering Services UK's number ranges 070/075/076 are associated with the so called *personal numbers* allocations [19]. We detail this specific category in the next section.

4.2.3.2 UK Personal Numbering Services

Personal Numbering Services (PRS) (also known as *international call forwarding services [44, 1]*) are premium numbers commonly used in information services or hospital lines. However, these numbers are often abused by fraudsters as part of scams or by deceiving a victim to call a number that charges higher cost than expected. As mentioned in 4.2.2, there were 34,424 unique phone numbers in UK range of `07x` PRS numbers, which were consistent with the allocation range of UK operators [20].

Many telecom operators, some of which are only virtual operators, offer the possibility to register such numbers online. These are often offered for free: the price of communications is shared between the registrant and the operator (often retaining between 30% and 50%). In addition to this, operators can forward incoming calls to international phone numbers. This can be used as anonymization service to hide the actual geographic location of the scammer.

An interesting observation is that certain operators are used more often than others to register scam numbers. Figure 4.1 shows the distribution of phone numbers used by scammers among the providers. We observe that, in our dataset, the top 4 operators (out of 88) provide more than 90% of fraud-related UK PRS numbers. In one case, fraud-related numbers represent almost 5% of an operator allocated numbers range.

By manually comparing those and other six operators [16], we found that scammers preferred operators that:

- Have an online registration and configuration service.

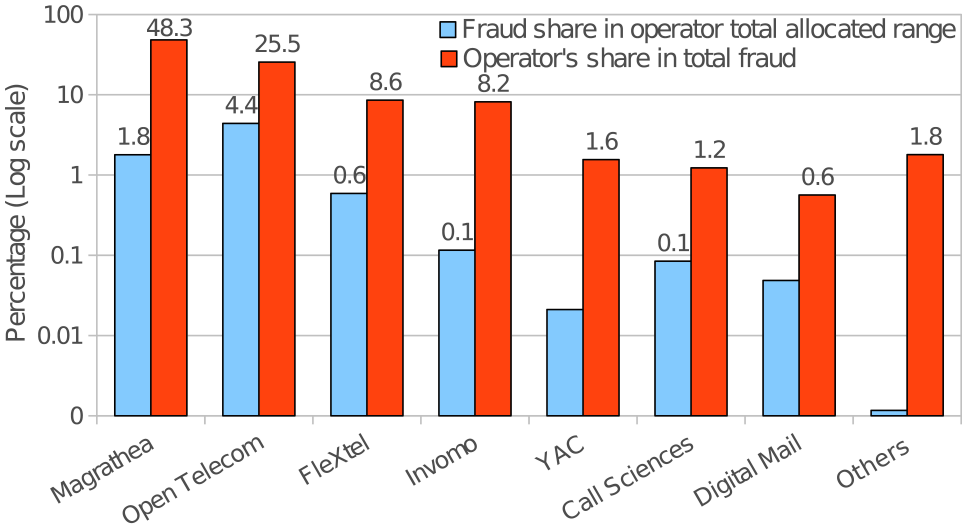


Figure 4.1: UK 07x fraud-share and fraud-vs-range allocation ratio.

- Provide an API to automate the registration process.
- Offer cheap or free international call forwarding.
- Offer a cash back program to pay the registrant for each incoming call.

Indeed, these features are appealing to scammers and, in general, cyber-criminals that perform illegal activities.

4.2.4 Criminals Behind the Phone

In this section, we used the SCAM dataset to evaluate the use of phone numbers to identify criminals, study their behavior, and unfold the structure and the size of their networks. Scammers are known to provide real phone numbers, at which they can be reached by their victims. Therefore, this dataset is less polluted with fake or spoofed numbers, which makes our results and conclusions more reliable.

4.2.4.1 The SCAM Dataset

The SCAM dataset covers the period from January 2009 to August 2012 (with the exception of August 2011, which is missing from our dataset [1]). For 16% of the phone numbers, we have the original email that was used to perpetrate the scam. These emails are classified in 10 categories, three of which cover over 90% of the data: *general scam* (62%), *fake lottery* (25%) and *next of kin* (inheritance) (8%).

A first look at the relation between phone numbers and scam categories shows that scams are not evenly distributed geographically. As shown in

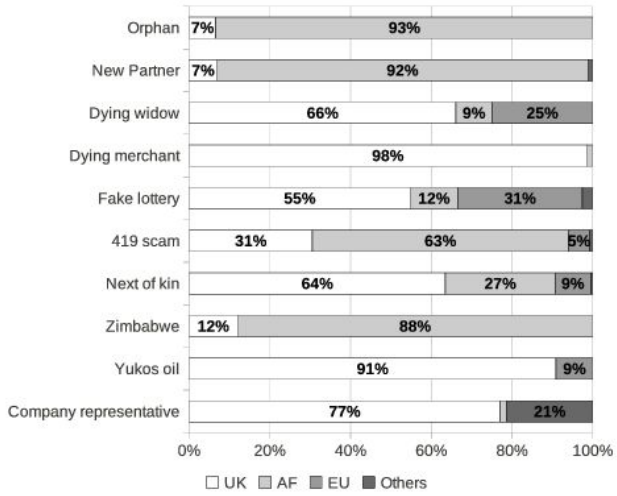


Figure 4.2: Scam email category preferences by phone number country codes.

Figure 4.2, certain types of scams rely mainly on African numbers (e.g., *new partner, orphan* scams), while others (e.g., *fake lottery, dying merchant, next of kin* scams) are almost always perpetrated by hiding behind a UK *personal number*.

4.2.4.2 Scam Communities

We first aimed at establishing relationships between phone numbers and email addresses used by scammers.

For this, we built a graph where the nodes represent either a phone number or an email address (that is used as point of contact in a scam message). The edges connecting the two types of nodes indicate that the owner of the address used that phone number in one of her scam emails. The initial graph has 34,740 nodes and 27,409 edges – 66% of nodes are emails and 34% are phone numbers. We then removed the smallest subgraphs (below 20 nodes) as they are less representative. We obtained 3,681 nodes (10.6%) and 4,360 edges (16%), consisting of 699 nodes as phone numbers and 2,982 nodes as email addresses. Globally, we identified 102 communities and 79 subgraphs.

The graph, a portion of which is shown in Figure 4.3, shows some interesting relationships. First, scammers seem to reuse a given email address to send scam messages, each message containing different phone numbers. Second, a given phone number seems to be reused in multiple scam messages or in combination with multiple different email addresses.

In particular, we observe that 37% of the phone numbers were reused by more than one scammer. Most of the largest nodes are white (phone

Figure 4.3: Visual relationships between phone numbers (white nodes) and email addresses (black nodes) that are used as point of contact in scam messages. The size of nodes is proportional to the number of edges.

numbers) and surrounded by several small black nodes (email addresses). This suggests that phone numbers play an important role in the activities of scammers. The set of phone numbers used by scammers in their campaigns is less diverse than the email addresses. In fact, email addresses are easily blacklisted and accounts are blocked when their connection with criminal activities is discovered. Also, while email addresses are virtually free, phone numbers are usually not. This forces the scammers to continually register fresh emails for new scam campaigns. Our analysis shows that phone numbers used in scams are more stable than emails and tend to be reused over time.

By looking at the smallest subgraphs, we notice that most of them contain phone numbers registered in a single country (76%), or a country combined with UK premium numbers (10%), originating mostly from UK, Benin or Nigeria. This indicates that most of the scammers work alone, or in small

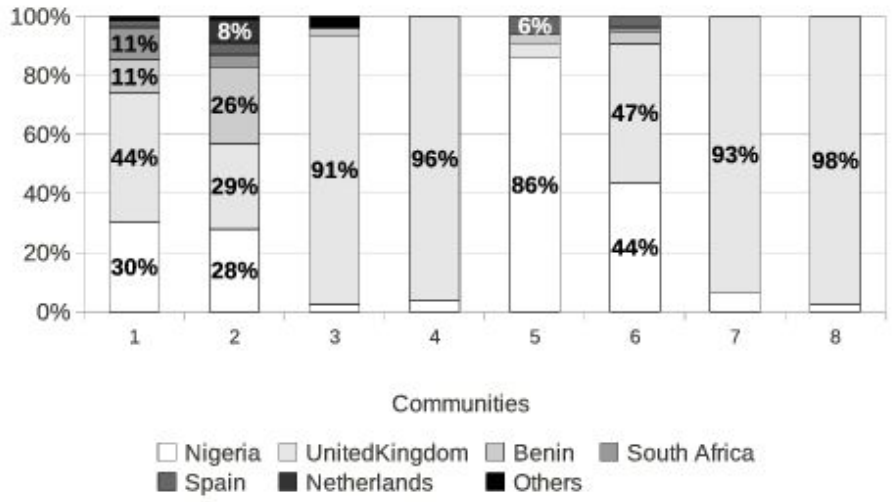


Figure 4.4: Top 8 largest communities in SCAM dataset, ordered by decreasing size from left to right.

Table 4.1: Count of SCAM phone numbers encountered in 2009-2011, reused in 2012. Includes all types of numbers.

Encounter year	Total numbers	Reused in 2012	%
2009	20,517	829	4%
2010	26,785	1,922	7%
2011	23,450	3,795	16%

groups located in a particular country. Figure 4.5 shows a real example of how scammers used four Spanish mobile phone numbers in the same campaign. All the email addresses are small variations of the same person's name, probably a character that the scammers tried to impersonate.

Looking at the largest communities - densely connected sets of nodes - we see that some groups are geographically distributed over several countries. For example, Figure 4.4 shows how the eight largest communities are organized. All these communities rely on UK premium numbers (for at least 29% of their phone numbers) and on numbers from Nigerian operators. Also, these communities use cellphone numbers in several European and African countries.

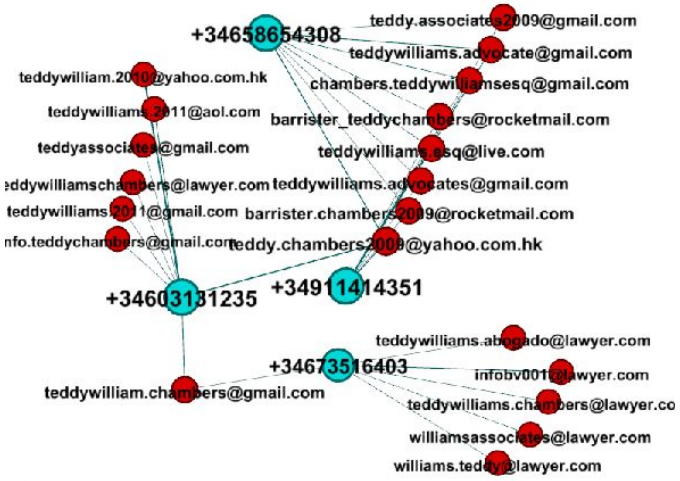


Figure 4.5: Example of links between phone numbers and email addresses.

4.2.4.3 Reusing Phone Numbers

We further tackle the question of reused phone numbers from a different angle. By looking at the SCAM dataset, which contains information on when these phone numbers have been used by the scammers (year and month), we understand that several of them were reused over long time periods.

Table 4.1 shows that 4% of the numbers that were in use in 2009 are still active in 2012. Figure 4.6 shows that as the period of time gets longer the amount of numbers being reused grows, from 21% (1 month) to 34% (3 months), and 48% over a year. In addition, a group of 307 phone numbers reappears yearly from 2009 to 2012. These figures do not include a detailed analysis of numbers reuse split by their type (e.g., UK PRS, mobile).

4.2.4.4 Discussion

The relationship between phone numbers and email addresses suggests two interesting findings. First, phones are more stable than emails and they are reused for longer periods. Therefore, phone numbers may constitute a better detection feature for the discussed threat categories. Second, even though the majority of scammers seem to operate in small groups, few communities appear to be spread over multiple countries.

However, this analysis alone is not enough to draw complete conclusions. For instance, we are still unsure how common is the phone number reuse habbit: given that 48% of phone numbers are reused within 12 months, does it mean that the remaining ones are discarded or does it mean that they are simply not reported by the website? Moreover, the fact that phones registered in different countries are used in conjunction with the

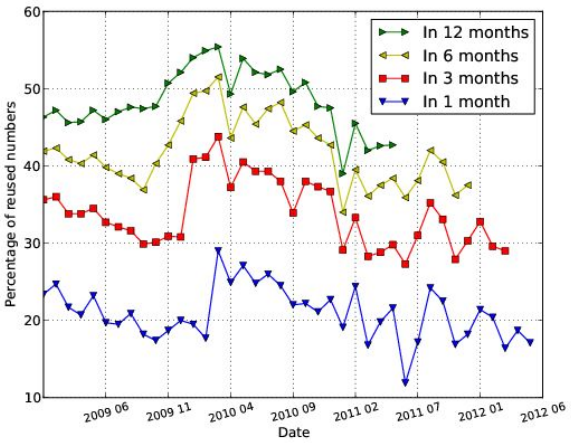


Figure 4.6: Accumulated shares of reused cellphones of scammers over time.

same email address might be the consequence of individuals owning multiple SIM cards (e.g., collected when traveling abroad). In the next section, we introduce a dynamic phone analysis technique that helps answering these questions.

4.2.5 Dynamic Analysis of Scam Phone Numbers

In order to understand the organization and the dynamics behind the scam communities identified in the previous sections, we performed periodic HLR lookups (Section 4.2.2) of the mobile phone numbers extracted previously. With this experiment, we aim at understanding how often mobile numbers are used in other countries (i.e., roaming) and over time.

Table 4.2: Mobile phone network status query results on 2012/08/02

Status	2012/01-06	%	2012/07	%
On the network	3,122	73%	984	84%
Replied with error	416	10%	67	6%
Turned off	734	17%	127	11%
Roaming	6	0.14%	3	0.26%

As we discussed previously, UK premium numbers (PRS) are often used by scammers to redirect calls, hiding the final call destination. We therefore had to exclude this category. We are left with 32,820 unique non-UK-PRS numbers out of which 29,685 are mobile phone numbers. Moreover, old numbers may be taken offline or assigned to a different customer. There-

fore, we eventually selected the 1,333 phone numbers that were collected recently (July-August 2012).

We verified that the selected two months period is representative of the general picture. To verify this, we performed a lookup on August 2nd, 2012 and compared the phone numbers reported in month of July 2012 with the phone numbers reported between January 2012 and June 2012. Table 4.2 shows that the population of mobile phones that were either reachable, roaming, or turned off is comparable in the two datasets, but more recently used phone numbers are more likely to be online at the time of our HLR query. This supports the fact that after a certain amount of time some phone numbers might be either discarded or replaced. Interestingly, very few numbers (only 9 in fact) were roaming in a foreign country. A first consideration is that mobile phone numbers are normally operated by criminals residing within their own countries, and not used while abroad or roaming.

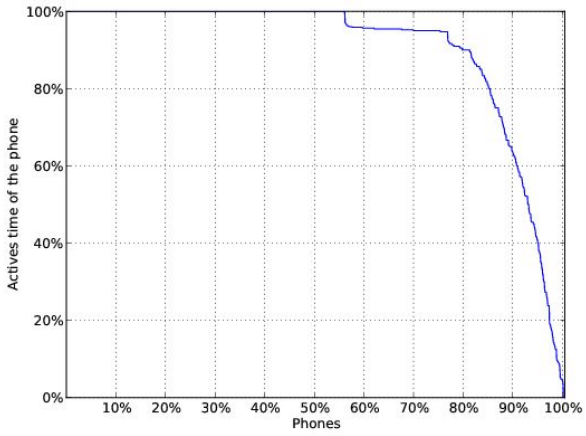


Figure 4.7: Mobile phone numbers sorted by frequency of OK status.

That is, our first experiment consisted of doing HLR lookups for the dataset of 1,333 recently used mobile numbers. We did queries every three days and for a period of two months. In order to appropriately choose this query window, we looked at how often the network status of a phone number is updated on average. A phone number first gets registered on the network and the HLR is updated instantly. When a phone gets turned off, the status is not updated, by default, but only when a call is received. By using one of our personal phone numbers, we determined the delay in a status change (e.g., from OK to OFF) as being 30 hours. Thus, a three days window seemed to be appropriate for our analysis.

By looking at changes in the network status attribute, we noticed that about half of the numbers have a constant OK status. This shows that scammers use phone numbers for long time periods by keeping them *online* most

of the time. It also means that they rarely switch to new phone numbers. In fact, only 97 phones appeared to be unregistered from the network for a long time (status `Absent Subscriber`). The overall distribution of the phone availability on the network is drawn in Figure 4.7. The average scammer keeps the phone switched `ON` most of the time and only 89 numbers were `OFF` more than 75% of the time. This appears to be in-line with the business model since scammers are interested in being reached by their victims.

Finally, according to the roaming status attribute, only 50 phones were used in a different country during our evaluation (i.e., roaming). The exact roaming locations are summarized in Figure 4.8. The Figure clearly shows two clusters – one in Africa and one in Europe – with a small intersection of the two. Nigeria is still a key country for this type of business, with about 80% of the roaming belonging to it. This again supports our hypothesis that distributed groups exist and that they operate coordinated and collaboratively from multiple countries.

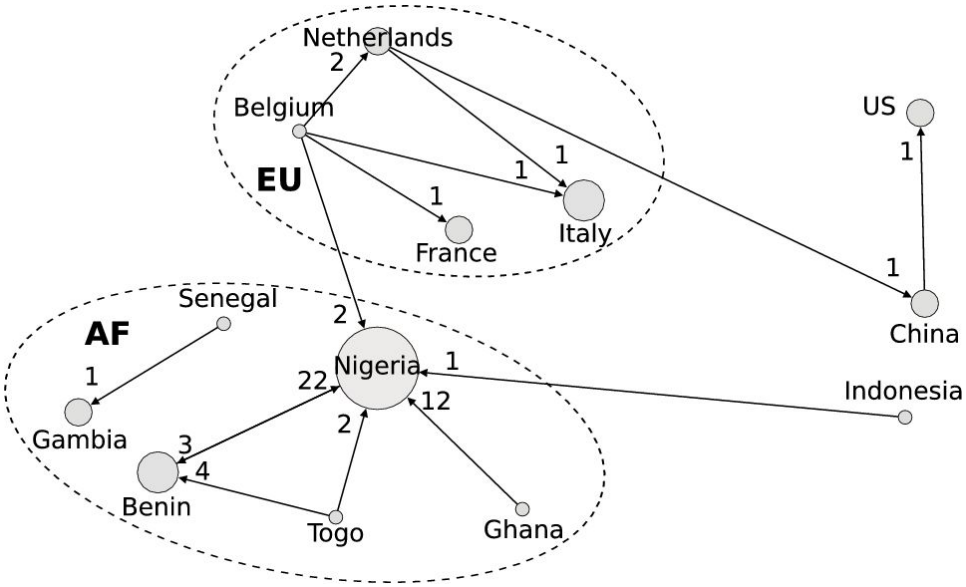


Figure 4.8: Mobile phones roaming per country. The arrow goes from the originating country to the roaming country. Edge labels indicate the number of roaming phones. The size of the node reflects the number of roaming phones in that country.

We then looked at the mobile operators, in order to evaluate if some of them are preferred over others. We analyzed the market share of the major four countries, which contain more than 700 numbers related to scam activities: Nigeria, Benin, South Africa and Senegal. Figure 4.9 shows the difference in distribution between the market share of each operator and the

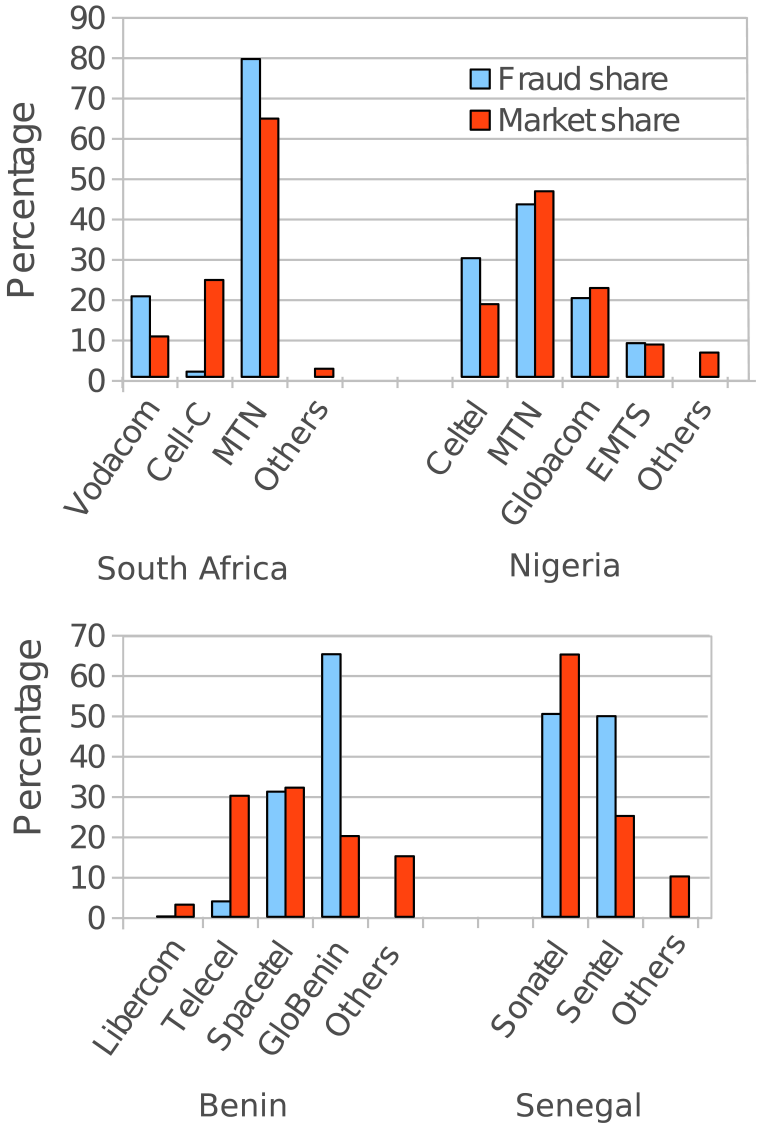


Figure 4.9: Distribution of mobile phone operators in Top 4 leading countries - market share vs. scam share.

"scam share" between criminals (dataset from December 2009 to December 2011). We can see that some operators seem to be less preferred by scammers (e.g., Cell-C in South Africa, Teracel in Benin), while others are clearly favored (e.g., GloBenin in Benin). The reason behind this might be due to pricing (e.g., for international calls) or stricter registration policies (e.g., strict ID checks). Like with UK PRS numbers we compared market-share and fraud-share of mobile network operators, however we did not notice any discrepancy between the two.

4.3 Automated collection and analysis of data on phone phishing

Phishers nowadays rely on a variety of channels, ranging from old-fashioned emails to instant messages, social networks, and the phone system (with both calls and text messages), with the goal of reaching more victims. As a consequence, modern phishing became a multi-faceted, even more pervasive threat that is inherently more difficult to study than traditional, email-based phishing.

In a previous work [84] we analyzed this type of scams, based on a selection of about 400 user-submitted reports, including the caller identifier (e.g., source phone number), (parts of) the transcribed conversation, general subject of the conversation, and spoken language. Besides confirming that vishing was popular in the U.S. at that time, our experience suggests that phishers rely on automated responders, and not only on live calls, with the goal of reaching a broader spectrum of victims. Reports were filed between 2009 and 2010 through a publicly-available web site where anyone can submit anonymous reports of vishing.

The system described in [84] focuses solely on vishing and, in addition, it has two main limitations. First, we trust submitters and, second, the effectiveness of vishing attacks could not be determined (evidently, people reporting suspicious calls are less prone to falling prey to them). To overcome these limitations, we propose to correlate the evidence on vishing scams with other forms of phishing. To this end, the new approach is to collect suspicious emails from spam-traps, instant messages from dedicated honeypots (e.g., based on myMSNhoneypot [30]) and content published by spammers on social networks (leveraging the @spamdetector service [116]). Our approach is content-driven. In particular, the first goal is to thoroughly quantify the popularity of voice-based scams. Secondly, we want to understand whether there are relationships between voice-based campaigns and text-based campaigns. Third, we strive to recognize evidence that suggest the use of social engineering techniques.

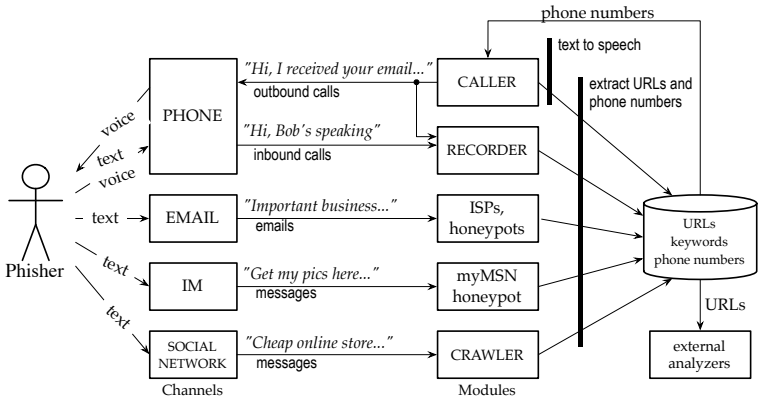


Figure 4.10: Overview of the dataflow of our collection system.

4.3.1 System overview

Our system has four modules, each tackling a different aspect of phishing. The *phone* module is an automated phone bot that places outbound calls, receives inbound ones, and records resulting conversations. The *email* module is a spam bot that receives spam and phishing email messages, and *IM* module is an instant messaging honeypot that collects unsolicited chat messages. The *social network* module will be implemented as a web crawler that to monitor suspicious accounts, known for sending spam (according to @spamdetector).

4.3.1.1 Text processing and correlation

The collected corpus (e.g., body of email messages, transcribed phone conversations, instant messages) is stored and analyzed using simple natural language processing techniques to extract popular sentences and words. Specifically, the stemming algorithm described in [99] is first applied to reduce words to stems. Secondly, stop words such as “the”, “an”, “this” are removed.

Regular expressions are then used to extract (possibly new) phone numbers and URLs. The former, core part of our approach, are sent to the phone module, while the latter will be shared for external analysis. Numbers, URLs and popular stems are used as a preliminary set of *features* to correlate messages across channels and find groups of different campaigns. Since shortened URLs are often used to evade filters (or simply to trick users), these are first resolved with the `long-shore.com` API, a service that mimics a real browser and records the redirection chain from a short URL to the target URL. Instead of the URL itself, the whole chain is retained and used as a similarity feature: it is indeed common for spammers to use multiple

redirections to the same phishing site, to increase the lifespan of their campaigns.

4.3.1.2 Phone channel

The core of our collection system is divided into two sub-modules, both based on Asterisk. The *caller* sub-module periodically calls a feed of numbers. Whenever someone answers, a pre-recorded prompt mimics a hypothetical victim, supposedly tricked by the reverse vishing scam (e.g., *"Hi, this is Bob, I received your email and I am curious to know more about it"*) and waits for 30 seconds. The resulting audio is recorded along with simple metadata such as date, time, and number. The *recorder* module is leveraged to answer inbound calls on a series of decoy numbers that we plan to make available deliberately on social network profiles, blog posts and forums.

Audio recorded from both inbound and outbound calls is retained in a database, and is transcribed using the Sphinx speech-to-text engine. The resulting text, if any, is then processed as described above.

4.3.1.3 Email channel

This module is implemented as a distributed client, meant to be deployed at ISPs and other institutions (e.g., universities and research centers). The client analyzes spam databases and collects emails that are likely to contain a phone number. At the moment, attachments that may contain scanned documents (used by scammers that attempt to evade basic filters) are not considered. Found messages are sent back to a bot, publicly reachable via SMTP at bot@phonephishing.info. Contributors are invited to submit suspicious emails directly to this address.

4.3.1.4 Instant messaging channel

This module is implemented as a set of instant messaging accounts (i.e., Yahoo! Messenger, Windows Live Messenger and Google Talk), all registered on myMSNhoneypot, a honeypot that monitors such accounts for any activity. Since the accounts all have empty buddy lists, any message or friendship request received on those accounts is considered as malicious. Only instant messages that contain phone numbers are retained.

4.3.2 Collected data

The email module has been tested for 2 months. To bootstrap the system, we gathered data from the email module and from phonephishing.info. We selected 551 vishing reports out from about a thousand of reports submitted by users in the first two years of activity. Discarded reports are mostly about

telemarketing calls. This may appear a limited amount of data, but it must be considered that people typically do not voluntarily give out information, especially when falling victims. Nevertheless, this module collected 532 *unique* numbers. We observed that a good share of the vishers resort to automated responders. In such calls, popular terms such as “press”, “credit”, “account”, are more frequent on automated calls with respect to calls made by live operators.

The email module has been processing spam emails provided from an ISP located in Southern California. In less than one month, the system selected 16,750 emails containing at least one telephone number, which amount to the 0.047% of the total number of spam emails collected by the ISP. Overall, this module collected 152 unique phone numbers as the time of writing.

With the support of a large telecommunication provider, the *phone* module is being deployed on a number of DSL lines to begin calling our initial list of 685 numbers.

4.3.3 Limitations and technical challenges

The main limitation of our approach lies in phone numbers collected by user-submitted reports, that could be very well spoofed identifiers. In fact, based on a few probing calls we placed manually, a good share of numbers (a rough 10%) are either deactivated or non-existing; unfortunately, it is difficult if not impossible to tell spoofed, blacklisted or deactivated numbers apart.

The main technical challenge of our system lies in the phone module. Specifically, even accurate speech-to-text software are far from being able of transcribing an entire conversation. We plan to workaround this obstacle by recognizing only a finite set of known (key)words extracted from reverse-vishing emails.

5

Social Network Forensics Framework

The use of online social networks and other digital communication services has become a prevalent activity of everyday life. Unfortunately, cybercriminals also became involved, and now use online social networks for fraudulent actions. As these services contain valuable information about one's activities, they could be used on crime investigations. In contrast to digital forensics that have standardized toolkits and procedures for evidence collection, crime investigators rely on custom techniques to acquire evidence from social networks. Moreover, the analysis and correlation of massive amounts of data scattered across diverse social networks is a challenging task.

In this chapter, we present a modular framework designed for assisting forensic investigators. First, it extracts the data from a user's social network profiles and other online services, taking advantage of stored credentials and session cookies. Next, it correlates user profiles across services, for providing a unified depiction of each user's activities. Finally, the visualization component, specifically designed for handling data representing activities and interactions in online social networks, provides dynamic "viewpoints" of varying granularity. The use of this framework could be deemed valuable towards analyzing incidents similar to "The Contact Dealer" scenario, presented on Deliverable 4.2. In particular, the framework could visualize connections with victims, and could potentially reveal interactions with selling parties (in cases where the Contact Dealer and selling parties exchanged mails or Skype chats)

5.1 The need for social network investigation tools

The term online social networks (OSNs) is commonly used for referring to services such as Facebook, Google+ and Twitter, narrowing the true extent of digital social networks. In reality, many more services can be construed

as OSNs, as they reflect sets of people with digital interactions. Digital interactions are created through the exchange of emails, VoIP calls, and a wide range of other every-day activities among individuals of a network. Thus, in the context of this chapter, with the term social networks we will implicitly refer to any online service that creates clusters of people with shared activities or communication, all of which can be sources of valuable information.

As the popularity and use of online social networks has increased, these services have become platforms for conducting nefarious activities such as scams or clickjacking attacks. The explosive growth rate of OSNs has, basically, created the first *digital generation* consisting of people of all ages and backgrounds. People are creating their digital counterparts for interacting with other users, for both recreational and professional reasons, and disclose a vast amount of personal data in an attempt to utilize these new services to the fullest. As a connection in a social network is a representation of social interaction, it also indirectly shows a level of trust between different individuals, in terms of the data they are willing to share with each other. However, the lack of technical literacy among the majority of users has resulted in a naive approach, where the caution demonstrated in social interactions of the physical world has disappeared.

This behavior has raised the concern of the research community in terms of user privacy. It also attracted fraudsters who found a fertile ground to target unwitting users. A wide range of threats exist, ranging from identity theft to monetary loss. While the amount of personal information disclosed by users [64] or leaked by services [79, 80] is troubling, in certain cases it can prove to have a positive "side-effect". Law enforcement agencies have been able to solve criminal cases after extracting the digital footprints of users, as they contained clues that ultimately led to the discovery of the perpetrators. Ideally, users will learn to be more privacy-aware, and limit the visibility scope of their personal information to a well-defined set of friends [24]. In such a scenario, when agencies *lawfully* acquire a suspect's device they will still be able to extract useful data from the accounts.

Social forensics tools aim to facilitate the discovery of this digital "trail of breadcrumbs", and extract data that can guide criminal investigations towards uncovering crucial information. Even though a multitude of digital forensics tools exist, they mostly focus on recovering deleted files or information from the device's volatile memory. The very few existing tools that target social networks tend to be proprietary commercial solutions. Our goal is to provide an extensive open source framework that will assist forensics analysts in this daunting task.

We have designed and implemented our toolset with the following usage model in mind: the authorities seize the digital devices (be it desktop, laptop

or just hard disk drives) of someone suspected for a crime or a fraud[1] and wish to acquire all the information regarding online activities. Social forensics analysis presents three major challenges: (i) acquiring as much data as possible from the suspect's online accounts and relevant local artifacts, (ii) correlating contacts across services, and (iii) visualizing this extensive collection of data. Our modular framework contains components for handling all three tasks.

The core functionality of any forensics analysis tool is the extraction of user data. We create a series of modules, each designed for extracting data from a specific service. When available, we take advantage of existing public APIs. In the remaining cases, we build custom crawlers for acquiring the data.

The correlation of users across services is a very crucial, yet challenging, aspect of our framework. Our correlation component follows a series of techniques for mapping user accounts from different services. Using a method we demonstrated in previous work [97], we map email addresses extracted from the suspect's accounts to Facebook profiles, which are the core sources of information. We conduct a similar process in Foursquare utilizing the search functionality of the official API. Furthermore, we employ data from `about.me`, a social directory site where users create a profile page with links to their social accounts, to further improve our correlation results. Finally, we also use fuzzy matching techniques for matching user names and email handles collected from different services.

The datasets collected during the data extraction process contain a wide range of different types of information regarding online activities. Existing forensics-related visualization tools usually focus on the depiction of graph-related data. However, various visualization libraries exist, and can handle multiple types of data. As such, we build upon existing libraries and create a visualization framework, designed specifically for visualizing data representing user activities in online social networks and communication services. Furthermore, the massive amount of data necessitates the creation of dynamic viewpoints of varying granularity, that will assist analysts in surveying aggregated statistics, as well as focusing on specific users or interactions. In summary, the contributions of this work are the following:

- We create an extensive framework for crawling a wide range of major social and communication services that, we hope, will set the foundations for the forensics community to build a complete and open source toolset.
- We leverage the search functionalities of social networks, a social directory and fuzzy matching, to correlate users across services and provide a unified view of a user's activities.

[1] For the remainder of this chapter, we will refer to this person as the *suspect* for reasons of simplicity.

- Our visualization framework provides perspectives of varying granularity, and enables users to dynamically shift focus to different aspects of user activities. This can facilitate filtering out the substantial amount of noise data available in forensics analysis.

5.2 Social Forensics

Digital forensics analysis has been a valuable asset in solving crimes in spite of its relatively young "age" compared to traditional forensics. Initially, the focus was on analyzing data stored on a computer, and recovering files that suspects had erased. However, as a result of the explosive advances of technology and its use creeping into all aspects of life, nowadays digital devices (e.g., laptops, smartphones) contain a significant amount of data that can assist the authorities in solving crimes.

A large amount of interaction takes place in online social networkings services and over digital communication media such as emails, instant messaging and VoIP networks. Users access information through these devices and save entries about their appointments in digital calendars. Furthermore, a large amount of data is saved online and not on a specific device. Thus, it is mandatory for forensics tools to extract data saved online, and not only extract data stored locally on a device. The goal of social forensics is to leverage social networking and communication services for extracting as much information as possible, regarding the online activities and communications of a suspect.

Multiple reports describe cases where the authorities have resorted to social networks for acquiring information, which has ultimately led to cases being solved (e.g., [87]). Even murder cases have been solved with the use of clues extracted from the suspect's digital communication and online activities [4, 59]. A survey held in 2012, among 600 law enforcement agencies from 48 states in the USA, reported that 92.4% of the agencies surveyed online social services [12]. For 77.1% this was done as part of criminal investigations. This survey reflects the significance of the data available in online services for assisting authorities in solving crimes. The importance of this information was also made evident by the case of PRISM[2], the electronic surveillance program operated by the United States National Security Agency (NSA), which was just recently made known publicly [11].

It is evident that the evolution of technology, and its widespread adoption in everyday life, mandates the evolution of investigating techniques as well. Thus, there is need for an extensive toolset that can extract data from all these services, correlate contacts and information across services, and provide visualization of the data in a dynamic and intuitive way.

[2]Discussing the ethics of such a program is outside the scope of this work.

Evaluating the completeness of a social forensics framework is complicated, due to the abundance of online services that exist (with new ones constantly emerging), each with its own properties, layout and format. This process is further complicated by the fact that a complete view requires data from other accounts as well (the suspect's contacts). Tools have to handle the restrictions set by service APIs, anti-crawl mechanisms and account privacy settings. Additionally, social networks frequently update their public APIs and change their layout, requiring modifications. Thus, the completeness of a social forensics tool has to be evaluated based on the *breadth* (i.e., the range of different services it can process), *depth* (i.e., the completeness of data) and "*up-to-dateness*" of the data extraction process.

Overall, our goal is to release an open source framework that targets a wide range of the popular services and extracts as much data as possible. Ideally, this will comprise the foundations for a large open source project that will attract contributors from the security community. Contributors can assist in expanding its breadth by creating modules for other services, its depth by implementing functions for accessing data that might not be attainable now, and keeping it up-to-date.

5.3 System Implementation

Our framework has been implemented in `Python` as a collection of components. We have designed it in a modular way so it can easily be extended by adding new modules for other social networks and services. In this section, we provide a high-level overview of our system, describe the role of each component, and present technical details regarding the implementation of some of the components we have created. Figure 5.1 presents the architecture of our framework and the steps that comprise the whole procedure:

1. The data collection component uses stored session cookies and user credentials to log into the online services as the suspect.
2. Each crawling component extracts as much data possible from each service that the user has an account for.
3. All extracted data is saved into a MySQL database.
4. The account correlator component:
   (a) Pulls the account information of the suspect's contacts from the database.
   (b) Uses several techniques for correlating the accounts, some of which leverage online services.
5. The data visualization component fetches data from the database asynchronously and dynamically presents the viewpoints requested.

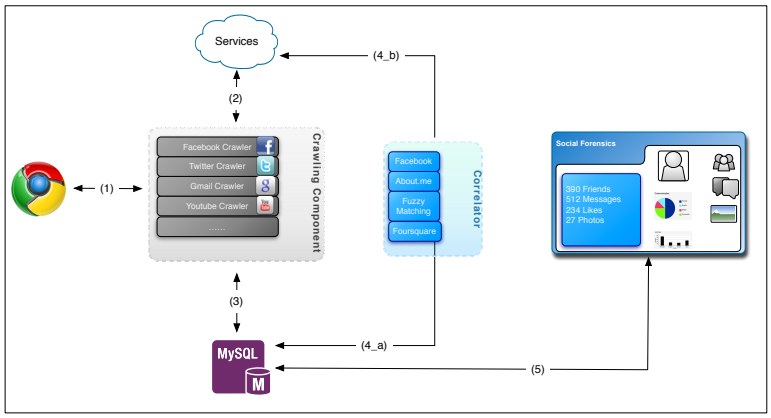


Figure 5.1: The architecture of our framework which is comprised of three major components.

5.3.1 Usage Scenario

We implemented our framework with the following usage scenario in mind. The forensics analysis investigator has acquired the suspect's digital device (or hard disk and connected it to a computer) and connected it to the Internet, since the data extraction and correlation components must connect to online services. Nonetheless, even though we have developed our system as a social forensics framework, it can also be useful in other situations. For example, users that want a unified view of their online activity history, with statistics regarding specific activities and per-user interactions, can also utilize our system.

An important design aspect of our system, was to make its execution as simple as possible. Ideally, the analyst would need only to execute a program and everything else would be done automatically. However, due to the requirement of authenticating the crawling modules that use public APIs with the social networks through `OAuth`, a small amount of manual intervention is needed. Specifically, after the system logs into a service, the investigator is prompted to authorize the crawling component for the suspect's profile (by simply clicking a button).

After the authorization phase, everything else is completed automatically. The framework installs a MySQL database and creates a series of tables for storing all the information from the suspect's accounts. The libraries required by the the crawling component, for example fbconsole [54] and Tweepy [18], are downloaded and installed automatically. The libraries for the visualization component are included within the web application.

5.3.2 Data collection components

Depending on the targeted service, the corresponding crawling component attempts to extract as much information as possible. In the case of online social services we leverage existing public APIs, if available. Otherwise we create custom crawlers for extracting the data. Here we provide technical details for certain modules.

Log-in process. Our tool uses the credentials saved in the browser's password manager or existing session cookies, to log into the targeted services as the suspect. Alternatively, the analyst can manually add the suspect's credentials in a configuration file, when no other method of logging in is available for a service.

The password managers of Chrome and Firefox utilize a `SQLite` database as their password manager back-end. Some browsers, like Firefox, retain this database encrypted using a "master password". On the contrary, the Chrome browser does not employ any encryption mechanisms, thus, storing the credentials in plaintext. We implemented a custom password extractor that locates Chrome's `SQLite` password file in the filesystem, and extracts credentials belonging to social networks and relevant services.

Browser session cookies are also stored in `SQLite` databases found locally in the filesystem. The same process is followed to extract session cookies.

Facebook. Once logged in, a custom application is installed in the suspect's profile, so the data can be retrieved through Facebook's Graph API [55]. This application has access to all resources available in the profile. After installation, our system leverages the Facebook Query Language (`FQL`) to extract the data from the user profiles [53]. `FQL` provides an `SQL`-like interface for querying user data, and can evaluate multiple queries in a single API call through `FQL` multiquery requests. Queries are packed as a `JSON`-encoded dictionary and sent as a single request. The response includes a similar dictionary with the respective results.

Twitter. In order for us to use the Twitter API, similar steps are followed. An application that has full access to the profile data has to be installed in the suspect's profile. Twitter poses an extra overhead during the crawling phase, due to its rate-limiting policy. Requests are performed with 10-second intervals, for avoiding potential rate-limiting issues. Protected accounts (whose information is only available to followers) are collected with the highest priority. Next, we focus on accounts with small volumes of data, i.e., those with the smallest volume of tweets.

Google+. We utilize the official Google Plus API [8] for extracting the data. Through this API and the `OAuth` authentication method, we are able to extract all the public information from the users' profiles, and the contacts from public circles. The information also includes the name of the city where

the user resides. We rely on the Google Geocoding API [10] for converting the city to a pair of geographical coordinates.

Foursquare. Our crawling component is built upon a `Python` wrapper [7] for the official Foursquare API [6]. After the `OAuth` authentication is completed and an authorization token is acquired, the crawler extracts the data through API calls that return the data formated as `JSON` objects.

5.3.3 Account correlation component

This component has a very important role. As our goal is to collect data from a multitude of online services, we require a method for correlating the suspect's contacts across services. Several separate modules comprise the component, each leveraging a different service or technique.

Facebook. in previous work [97] we demonstrated how Facebook can be leveraged as an oracle for mapping a user's email address to his online account. We follow this technique for mapping all email addresses of users, that have correspondence with the suspect to their Facebook accounts (if they have created one with that email address). While a user can change the privacy settings to be removed from such searches, it is enabled by default and too complex for average users to disable.

Foursquare. The official API contains a call that searches for Foursquare accounts based on different types of information and can, thus, also be used as an oracle for correlating user accounts. Specifically, the API call takes as a parameter any one of the following pieces of information and returns the relevant Foursquare account (if it exists): Facebook ID, Twitter handle, email address, name, phone number. Thus, apart from locating a user's Foursquare account, we can also associate disjoint pieces of information we have collected from other services.

About.me. This site offers a platform for users, where they can create a personal page that contains links to their accounts on popular social networking services. Using the names extracted in previous steps, we search for `about.me` profiles with the same name and extract the links to their profiles on social services. We then attempt to verify that the account belongs to the same user by comparing the account IDs to any we have correlated previously.

First we leverage the website's search functionality for locating the suspect's contacts that have an `about.me` profile. As the search query results are dynamically rendered through `Ajax` requests, we scrape the results through `PhantomJS` [15], a headless webkit that also offers a `Javascript` API. After obtaining the user profiles, we extract the available links towards social network profiles. Each link to a specific `<network>` is accessible through a unique URL[3].

[3] `http://about.me/content/<username>/<network>`

Fuzzy matching. Several of the services we extract data from don't provide the email addresses of the account's contacts, which would allow us to deterministically correlate user accounts across services. To overcome this, we compare user names across services and match them based on similarity. While this method follows a "fuzzy" approach, we are able to obtain results, as users tend to reuse user names across services, or simple variations of them. For example, a user with a Facebook profile under the name "John Doe" might have an email address handle "`john_doe`", "`johndoe80`" etc. This module also creates synthetic email addresses using certain variations of the user name (e.g "`john_doe`", "`doe_john`") along with the most common email providers (namely "`gmail.com`", "`yahoo.com`", "`hotmail.com`", "`windowslive.com`", "`msn.com`") and passes them to the other modules.

User input. While the above methods yield results, nonetheless, certain accounts may not be correlated with others belonging to the same user. This could be due to users creating multiple accounts under completely different user names. As this correlation can provide invaluable information during the visual inspection of the data by analysts, our visualization component enables the manual correlation of accounts. Specifically, the user can correlate an account from one service with accounts from other services. That information is saved, and the dynamic perspectives will reflect the new associations. Similarly, the user can remove any erroneous correlations made during the automatic correlation procedure by the fuzzy matching module

5.3.4 Visualization components

Our goal is to develop a modern visualization platform that will offer a wide variety of graphic data representations, while remaining portable. This led us to create it as a web application. The front-end is designed to run on the same machine where the data is kept.

The vast amount of data mandated the use of an asynchronous, event-driven model for the front-end, where data is fetched upon request. The front-end is built upon `AJAX` requests using the `jQuery` framework [13] for data retrieval and manipulation.

The ever-growing need for complex data visualization has lead to the release of powerful frameworks. `D3.js` [3] is a JavaScript visualization library capable of rendering a variety of schematics such as Graph layouts and Calendar Views among others. This framework is used for the majority of visualizations incorporated in the front-end. Moreover, we leverage the Google Maps JavaScript API [9] to render location-based information, when available, on a map.

5.4 Data Collection

In this section we present a list of the services from which we collect user data, as well as a description of the types of information acquired. For every online social network, we also collect any information that is reachable for every one of the suspect's contacts.

Facebook. This is the main source of information, as it is the most popular online social network, and users tend to reveal a large amount of personal information on it. Our crawling component extracts any of the following information that exists:

- *Personal information*: this may include current location, hometown, education and work information.
- *Contact list*: apart from the list of the suspect's contacts, we also collect any custom lists and the contacts contained in each list.
- *Status updates* and any links contained.
- *Chat logs* along with timestamps for each message.
- *Photos:* links to the photos and information regarding photo albums, photo timestamps and tagged users.
- *Videos* uploaded by the suspect, and videos he has been tagged in.
- *Check-ins:* the places the suspect has checked into, the timestamp and the data and coordinates of the place, along with tagged users.
- *Likes:* activities and articles the suspect has liked.
- *Shares:* pages the suspect has shared.
- *Fan pages* (also checks if the user is an administrator of the page).
- *Events* and the information of the users that participated.
- *Groups* the suspect is a member of, and the information of the other members.
- *Notifications* the suspect has received.
- *User notes*.
- *Contact information:* we also collect all of the aforementioned data from the contacts that is viewable through the suspect's account (e.g., a contact's chat messages are not viewable.)

Twitter. We first collect the account's information and contact list. That includes the accounts the suspect follows as well as those following the suspect. We also collect the suspect's tweets as well as any tweets re-tweeted, and all available metadata (e.g. timestamps, location).

Foursquare. We collect the suspect's check-ins along with the corresponding metadata. Specifically, we collect the timestamp, the venue's name,

VenueID, and location coordinates. We also collect the list of friends, and any links to their profiles on other networks. Unfortunately, due to limits set by the API and website, we can only retrieve the last 100 check-ins of the suspect's friends.

Skype. We first collect the list of contacts and their disclosed information (which may include location, gender, date of birth). Then we extract the history of chat logs and relevant metadata, as well as call history (and duration) and file exchanges. We also attempt to retrieve any exchanged files that are still located on the hard drive.

Gmail. We collect all emails exchanged with the suspect, and extract the email addresses and any names associated with those addresses. For each email we also collect the relevant metadata.

Google. We access the suspect's account in Google and extract the relevant information from Google calendar and Google Docs. Specifically, we collect all calendar entries (which may contain a location, a description, and other users attending), and download documents accessible (we also retrieve information about which other contacts have access to the documents).

Google+. We first collect the suspect's contacts contained in the various "circles" (i.e. contact groups), and the suspect's activities; posts, comments, shares, and "+1"s (similar to likes in Facebook). We extract publicly available data from the accounts of the contacts, as well as any accounts that have commented on the suspect's profile (even if they are not part of one of his circles).

Youtube. We first collect the suspect's information. Then we extract the history of watched videos, and channel subscriptions, playlists, uploaded videos and their comments and favorited videos.

Dropbox. We first locate the Dropbox folder, depending on the suspect's operating system, by retrieving the information from the application data. Then, by traversing the Dropbox directory tree, we extract all the files with their corresponding metadata. We also keep the application data that can be used for other aspects of forensic analysis [5].

5.5 Activity Visualization

In this section we describe the various methods for visualizing our collected data. The plethora of services that can be used by suspects require a grouping of this disjoint information into a unified set, where actions across services are correlated (e.g. what type of communication does the suspect have with user X from all services). Furthermore, the abundance of available information necessitates the ability to shift focus to specific activities (e.g., status updates on Facebook), and interactions (e.g., users with the largest amount of shared activities with the suspect). Thus, we provide the analyst

(a) User granularity (b) Service granularity

Figure 5.2: Two elements from the aggregated statistics perspective. We provide details regarding the most interesting activities, at a user-granularity and service-level granularity.

with dynamic "perspectives" of varying granularity, with aggregated correlations as well as fine-grained views of the collected data. We have several viewpoints for creating the different perspectives.

Aggregated. Here we present aggregated statistics regarding the most interesting activities from all the services. With one glance, the analyst can see which services the suspect mainly uses, and what data is available. In Figures 5.2(a) and 5.2(b) we see an example screenshot regarding some of the aggregated statistics presented in this viewpoint. Specifically, we can see the most important types of data across services and a more detailed description of activities per service, respectively.

Service. Here we focus on a specific service, and present aggregate statistics regarding the users activities. A list presents the contacts that have had the most communication with the suspect. Next, as shown in Figure 5.3, we depict the structure of the social graph and the interconnections between all contacts. The node's size is based on the number of connections the contact has. Thus, the analyst can immediately recognize heavily connected users or outliers. The graph can plot contacts of a specific service as shown here, or a combined view of all services where the contact's of each service have a common color. Each graph node represents a user, and when clicked presents the contact's name and photo. Furthermore, a contact search function dynamically detects and highlights nodes in the graph, allowing investigators to quickly identify contacts of interest inside the graph.

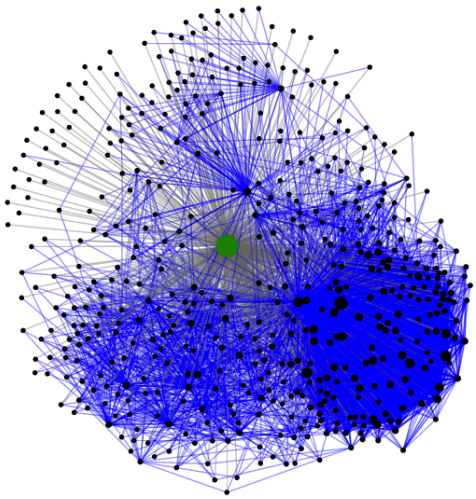


Figure 5.3: An example plot of the suspect's social graph. The suspect is depicted with the green node. The size of a node is defined by its degree of connectivity. Edges toward the suspect's node are grey, while edges between contacts are blue.

In Figure 5.4 we present a screenshot of a graph that visualizes the total communication between suspect and online contacts. The amount of shared activity defines the width of the connector. This enables the users with the most communication to be easily identified and scrutinized. When the connector is clicked, a window presents all the shared activities.

User. A very important viewpoint is that which focuses on a specific user. Once the analyst has identified online contacts that might be of interest, he can use one of two perspectives. First, one can select the contact and be redirected to an aggregated statistics viewpoint, containing all information available regarding the actions of that contact across all services (based on the number of accounts that have been associated during the correlation phase). Second, the analyst can choose to focus only on the shared activities the contact has with the suspect across all services. That includes, chat messages, emails sent, wall posts, shared photos, etc. The coarse-grained perspective presents aggregated statistics, while the more fine-grained perspective allows to focus on a specific type of activity. In both viewpoints, the investigator can ultimately view all individual activity and communication resources, e.g., exchanged messages, pages "liked", or Skype calls. Furthermore, the viewpoints can be dynamically configured to visualize data from one or all services.

Timestamp. A very important factor for visualizing relevant data, is that of time. Every perspective contains a color-coded calendar depicting the amount of activity a user has conducted on a specific date. We show a

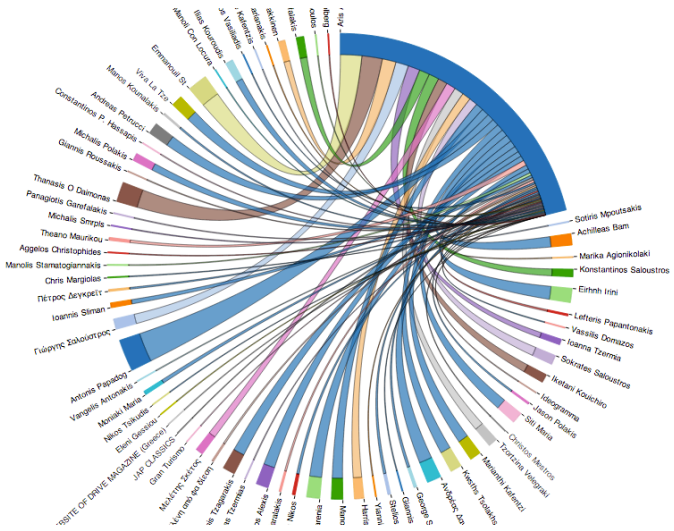


Figure 5.4: This graph plots the overall communication between the suspect and his contacts. This visualization facilitates the recognition of important contacts, as the volume of communication determines the width of the connection between suspect and contact.

segment of an example calendar in Figure 5.5, where blank squares depict days where the user did not send or receive any emails, while the red square depicts a day with a large volume of exchanged emails. However, the analyst might wish to focus on the activities of the suspect during a specific time period which is of interest. As such, certain viewpoints can dynamically change and allow one to focus on a specific time window.

Content. A word cloud provides the analyst with a quick view of the most common words contained in the suspect's communication, which can be across services or focused on a specific service or user. Thus, recurring motives and topics can easily be spotted. In the case of Twitter, we also create a word cloud with the hashtags (i.e., topics) of the suspect's tweets as we show in Figure 5.6. This can also reveal subjects that the suspect tends to follow or comment on (e.g. politics, religion) and can be relevant to the analyst's investigation. Clicking on one of the terms will fetch all the messages, emails, tweets or posts containing the term.

Location. A very important piece of information is the suspect's location. Using information from the suspect's check-ins and residence we plot a map with the locations he has visited, and also visually annotate the amount of times each location has been visited. Furthermore, the analyst can also define a time window, within which all of the suspect's activities are correlated with that location. For example, with a time window of one hour, by clicking on the location marker, a window will inform of all the activities (e.g.

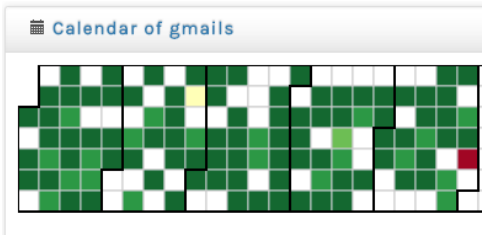


Figure 5.5: Extract of the calendar element, depicting the email exchange activity of the suspect over a period of five months.

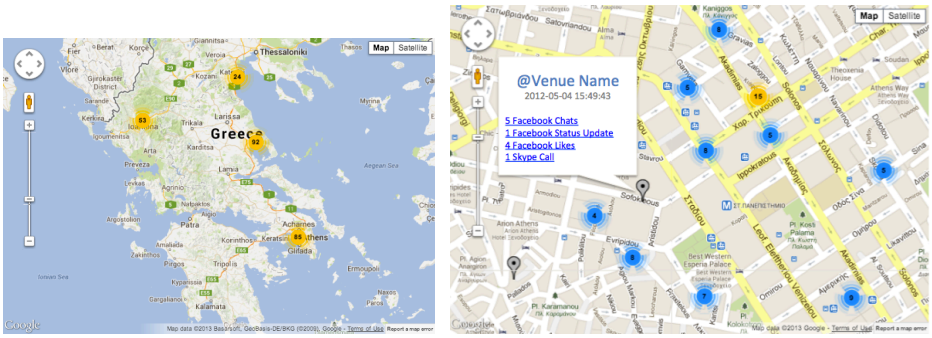
Figure 5.6: The word cloud shows the words most frequently contained in the suspect's communications. Here we see an example created from a user's Twitter hashtags (topics).

chat, Skype calls) the suspect conducted up to one hour after the check-in. Thus, the analyst can associate important activities to specific locations or even search for patterns of activities at certain locations.

In Figure 5.7(a) we can see an example screenshot showing the aggregated check-ins at a city-level granularity. Figure 5.7(b) depicts a closer view of a specific region, with the information window for a specific check-in. The window presents the name of the venue, the check-in timestamp and a series of activities that have been completed within a one-hour time window. All elements are click-able for presenting the resources of interest.

Furthermore, as a specific period might be of interest, we can plot the check-ins conducted during a specific period of time. Also, the investigator can select a contact, a distance X and a time duration T, and the map presents any check-ins that the suspect and the contact conducted with a time difference up to T at venues that have a maximum distance of X.

Photographs. A valuable resource of information in criminal investigations are photos found in social networks, as demonstrated in the case of the



(a) Aggregated view (b) Close-up view

Figure 5.7: Two views of the map plotting the suspect's check-ins. (a) An aggregated city-level view. (b) The details of a specific check-in and the associated activities.

Vancouver riots [87], where vandals were identified through photos posted in social networks. The investigator can select to view all the photos collected from the suspect's profiles. Any available user tag information is also presented, and statistics show the contacts with the most common photos with the suspect (based on tag information).

6 Conclusions and future works

In this deliverable, we reported on several recent research directions and on the state of the art in the analysis of Internet-related fraud activities.

We started by analyzing, in Chapter 2, the complex phenomenon of Internet banking and credit card fraud, the current approaches to perform anomaly detection and fraud analysis. We identified several shortcomings and research directions which the partners may wish to investigate in the remainder of the project.

In Chapter 3 we discussed the specific sub-threat of information stealing trojans, and we reported on two approaches we developed to analyze them.

P2P Zeus is a significant evolution of earlier Zeus variants. Compared to traditional centralized versions of Zeus, P2P Zeus is much more resilient against takedown attempts. Potential countermeasures against P2P Zeus are complicated by its application of RSA-2048 signatures to mission critical messages, and rogue bot insertion is complicated by the Zeus message encryption mechanism which makes the use of random bot identifiers impossible. Poisoning attempts are forced to use widely distributed IPs due to a per-bot IP filter which only allows a single IP per /20 subnet. The network's resilience against takedown efforts is further increased by its use of a Domain Generation Algorithm backup channel, and by an automatic blacklisting mechanism. P2P Zeus demonstrates that modern P2P botnets represent a new level of botnet resilience, previously unseen in centralized botnets.

Additionally, we presented Zarathustra, an automated system to observe on the client side the activity of banking trojans that perform WebInjects. Zarathustra generates signatures of the DOM differences by comparing web pages as they are rendered in an instrumented browser running on clean and infected virtual machines. It first builds a model of legitimate differences, and then builds a signature of the modifications introduced by the malware sample. Our system has the advantage of requiring no reverse-

engineering effort: The only requirement is a binary sample of the malware. Signature generation is completely automated and independent from the malware family. In this sense it is a major improvement over the current state of the art.

Our evaluation of Zarathustra against 213 real, live URLs of banking websites and 56 distinct samples of ZeuS shows that, in all the cases, our system extracted all the injections correctly. The low rate of false positives (1.0%) were caused by legitimate differences in the original web pages. We have developed specific heuristics, which can be safely enabled under realistic conditions, that can further reduce such false positives to zero. Zarathustra scales well, and can generate fingerprints for 1 URL in less than 3 seconds on average even on our modest infrastructure.

Although simple, our approach has the great advantage of being completely agnostic with respect to the source of the differences: As long as the manipulated data is observable, our approach can be generalized to create further "difference modeling" techniques that can be used to characterize the activity of an information stealer from other observation points.

In Chapter 4 we analyzed the role of phone numbers in cyber-crime schemes.

First, using nigerian scams as a case study, we collected a number of datasets and designed a technique to identify and extract phone numbers out of them. A first result is that extracting phone numbers from unstructured text is challenging and inaccurate with current tools.

We then focused on analyzing the role of phone numbers in scam related frauds. We identified different groups, created strong links between apparently unrelated actors and analyzed their geographic distributions.

While a phone number appears to be a weak metric for identifying spam messages, on scams messages it proved to be a good identification mechanism when compared to email addresses. We showed that this may be helpful in analyzing scammers operations, possibly supporting investigations in order to reduce future scam messages. The reuse of phone numbers is vital in certain business models where trust must be established over a long period of time (e.g., wire funds transfer fraud). For other business models, changing the phone numbers for cyber criminals might be more vital for their untraceability. One option is to change the SIM cards, but it requires operational risks (e.g., ID checks) and other overheads. Another option is to use virtual mobile numbers (VMN). VMNs are most inviting, with competitive or free pricing, laxed ID checks, and most importantly with remote operation and high-level API automation.

We discussed common business models found during our experiments. Our results show that a restricted number of mobile operators are used to deliver the majority of fraud related numbers. This suggests that some operators are preferred over others by fraudsters.

We are now developing a data collection system to capture different aspects of phishing campaigns, with a particular focus on the emerging use of the voice channel. The general approach is to record inbound calls received on decoy phone lines, place outbound calls to the same caller identifiers (when available) and also to telephone numbers obtained from different sources. Specifically, our system analyzes instant messages (e.g., automated social engineering attempts) and suspicious emails (e.g., spam, phishing), and extracts telephone numbers, URLs and popular words from the content. In addition, users can voluntarily submit voice phishing (vishing) attempts through a public website. Extracted telephone numbers, URLs and popular words will be correlated to recognize campaigns by means of cross-channel relationships between messages.

Finally, in Chapter 5 we addressed the growing importance of data found in online social network profiles for solving criminal investigations, and presented a modular framework that targets popular online social networks, and consists of components that perform three distinct tasks. First, all data that is reachable from the suspect's profiles is extracted, including the activities of contacts. Second, the contact profiles from different services are correlated, for creating abstracted profiles that contain a user's activities regardless the service of origin. Lastly, our visualization framework provides perspectives that focus on different types of data, and can dynamically change their level of granularity, shifting from aggregated statistics to detailed information. Nonetheless, the continuous evolution of these services requires a dedicated community for expanding and maintaining a complete toolset. Our goal is to release our framework and hope it will attract the collaboration of other members of the community, for maintaining a complete and up-to-date social forensics framework. An example where this framework can be utilized is the "The Contact Dealer" scenario, described on Deliverable 4.2. The framework can be used to track actions of the suspect and shed light to interactions with malevolent parties.

Bibliography

[1] 419 Scam Fraud Directory. http://www.419scam.org/419-by-phone.htm.

[2] Bulk SMS services and HLR lookups. http://routomessaging.com/.

[3] D3.js - Data-Driven Documents. http://d3js.org.

[4] Department of Law, State of New Jersey. Melanie McGuire found guilty of murder in 2004. http://www.nj.gov/oag/newsreleases07/pr20070423a.html.

[5] Forensic focus: Dropbox forensics. www.forensicfocus.com/Content/pid=429/page=2/#database.

[6] Foursquare API Endpoints. https://developer.foursquare.com/docs/.

[7] Foursquare wrapper. https://github.com/mLewisLogic/foursquare.

[8] Google+ API. https://developers.google.com/+/api/.

[9] Google Developers - Google Maps JavaScript API v3. https://developers.google.com/maps/documentation/javascript/.

[10] Google Geocoding API. https://developers.google.com/maps/documentation/geocoding/.

[11] The guardian: NSA prism program taps in to user data of apple, google and others. http://www.guardian.co.uk/world/2013/jun/06/us-tech-giants-nsa-data.

[12] IACP center for social media, 2012 survey results. http://www.iacpsocialmedia.org/Resources/Publications/2012SurveyResults.aspx.

[13] jQuery. http://jquery.com.

[14] Locating mobile phones. http://events.ccc.de/congress/2008/Fahrplan/attachments/1262_25c3-locating-mobile-phones.pdf.

[15] PhantomJS: Headless WebKit with JavaScript API. http://phantomjs.org/.

[16] Premium Rate Services Network Operators Contact. http://www.phonepayplus.org.uk/For-Business/Setting-up-a-premium-rate-service/Network-operator-contacts.aspx.

[17] The Koobface malware gang exposed. http://www.sophos.com/medialibrary/PDFs/other/sophoskoobfacearticle_rev_na.pdf.

[18] tweepy. https://github.com/tweepy/tweepy.

[19] UK Ofcom Numbering Site. http://www.ofcom.org.uk/static/numbering/index.htm.

[20] UK Phone Info Codes Allocations Lookup. http://www.ukphoneinfo.com/s7_code_allocations.php?GNG=70.

[21] Worldwide National Numbering Plans Collection. http://bsmilano.it/.

[22] Reversal and Analysis of Zeus and SpyEye Banking Trojans. Technical report, IOActive, 2012.

[23] State and trends of the "russian" digital crime market 2011. Technical report, Group IB, 2012.

[24] F. Adu-Oppong, C. K. Gardiner, A. Kapadia, and P. P. Tsang. Social circles: Tackling privacy in social networks. In *Symposium on Usable Privacy and Security (SOUPS)*, 2008.

[25] V. Aggelis. Offline Internet Banking Fraud Detection. In *ARES*, pages 904–905. IEEE Computer Society, 2006.

[26] E. Aleskerov, B. Freisleben, and B. Rao. CARDWATCH: a neural network based database mining system for credit card fraud detection. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pages 220–226, 1997.

[27] R. Anderson. *Security Engineering*. John Wiley & Sons, 2008.

[28] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99)*, pages 49–60, Philadelphia, PA, 1999.

[29] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In *Proceedings of the 21st USENIX Security Symposium*, Bellevue, WA, USA, 2012.

[30] S. Antonatos, I. Polakis, T. Petsas, and E. P. Markatos. A systematic characterization of im threats using honeypots. In *NDSS*, 2010.

[31] AV-Test.org. http://www.av-test.org/en/statistics/malware/.

[32] E. L. Barse, H. Kvarnström, and E. Jonsson. Synthesizing Test Data for Fraud Detection Systems. In *ACSAC*, pages 384–394. IEEE Computer Society, 2003.

[33] S. D. Bay and M. J. Pazzani. Detecting Group Differences: Mining Contrast Sets. *Data Mining and Knowledge Discovery*, 5(3):213–246, July 2001.

[34] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the Analysis of the Zeus Botnet Crimeware Toolkit. In *Proceedings of the 8th Annual Conference on Privacy, Security and Trust*, Ottawa, Ontario, Canada, August 2010.

[35] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *Privacy Security and Trust*, pages 31–38. IEEE, 2010.

[36] R. Bolton and D. Hand. Peer group analysis. Technical report, Imperial College, 2001.

[37] R. J. Bolton and David. Statistical fraud detection: A review. *Statistical Science*, 17, 2002.

[38] A. Brabazon, J. Cahill, P. Keenan, and D. Walsh. Identifying online credit card fraud using Artificial Immune Systems. In *IEEE Congress on Evolutionary Computation*, pages 1–7. IEEE, 2010.

[39] R. Brause, T. Langsdorf, and M. Hepp. Neural Data Mining for Credit Card Fraud Detection. In *ICTAI*, pages 103–106, 1999.

[40] A. Buescher, F. Leder, and T. Siebert. Banksafe information stealer detection inside the web browser. In *RAID '11*, pages 262–280. Springer, 2011.

[41] A. Cansado and A. Soto. Unsupervised Anomaly Detection in Large Databases Using Bayesian Networks. *Applied Artificial Intelligence*, 22(4):309–330, 2008.

[42] L. Cao, H. Zhang, Y. Zhao, D. Luo, and C. Zhang. Combined Mining: Discovering Informative Knowledge in Complex Data. *Trans. Sys. Man Cyber. Part B*, 41(3):699–712, June 2011.

[43] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41:15:1–15:58, July 2009.

[44] N. Christin, S. S. Yanagihara, and K. Kamataki. Dissecting one click frauds. CCS '10. ACM, 2010.

[45] D. Cook, J. Hartnett, K. Manderson, and J. Scanlan. Catching spam before it arrives: domain specific dynamic blacklists. In *Proceedings of the 2006 Australasian workshops on Grid computing and e-research*, volume 54 of *ACSW Frontiers '06*, 2006.

[46] T. Cymru. the underground economy: priceless. http://www.usenix.org/publications/login/2006-12/openpdfs/cymru.pdf, December 2006.

[47] D. Dagon, G. Gu, C. P. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Proceedings of the 23rd Annual Computer Security Applications Conference*, 2007.

[48] D. Dittrich and S. Dietrich. P2P as Botnet Command and Control: A Deeper Insight. In *Proceedings of the 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, October 2008.

[49] G. Dong, X. Zhang, L. Wong, and J. Li. CAEP: Classification by Aggregating Emerging Patterns. In *Proceedings of the Second International Conference on Discovery Science*, DS '99, pages 30–42, London, UK, UK, 1999. Springer-Verlag.

[50] E. Edelson. The 419 scam: information warfare on the spam front and a proposal for local filtering. *Computers & Security*, 22(5), 2003.

[51] A. Emigh. The crimeware landscape: Malware, phishing, identity theft and beyond. *J. Digital Forensic Practice*, 1(3), 2006.

[52] E. Eskin. Anomaly Detection over Noisy Data using Learned Probability Distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 255–262, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[53] Facebook. Facebook Query Language (FQL) Reference. https://developers.facebook.com/docs/reference/fql/.

[54] Facebook. fbconsole. https://github.com/facebook/fbconsole.

[55] Facebook. Graph API. https://developers.facebook.com/docs/reference/api/.

[56] N. Falliere and E. Chien. Zeus: King of the Bots, 2009. Technical Report, Symantec.

[57] W. Fan, M. Miller, S. Stolfo, W. Lee, and P. Chan. Using artificial anomalies to detect unknown and known network intrusions. *Knowl. Inf. Syst.*, 6(5):507–527, Sept. 2004.

[58] T. Fawcett and F. Provost. Activity Monitoring: Noticing interesting changes in behavior. In *Proceedings on the SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.

[59] Forbes. Solving a teen murder by following a trail of digital evidence. http://www.forbes.com/sites/kashmirhill/2011/11/03/solving-a-teen-murder-by-following-a-trail-of-digital-evidence/.

[60] A. K. Ghosh and A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In *Proceedings of the 8th conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 12–12, Berkeley, CA, USA, 1999. USENIX Association.

[61] M. Goncharov. Russian underground 101. Technical report, Trend Micro Inc., 2012.

[62] C. Grier, L. Ballard, J. Caballero, N. Chachra, C. J. Dietrich, K. Levchenko, P. Mavrommatis, D. McCoy, A. Nappa, and A. Pitsillidis. Manufacturing Compromise: The Emergence of Exploit-as-a-Service. In *ACM conference on Computer and Communications Security*, 2012.

[63] C. Grier, K. Thomas, V. Paxson, and M. Zhang. @spam: the underground on 140 characters or less. In *Proc. of the 17th ACM conf. on Computer and Communications Security*, CCS '10, pages 27–37, New York, NY, USA, 2010. ACM.

[64] R. Gross and A. Acquisti. Information revelation and privacy in online social networks. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05', pages 71–80, New York, NY, USA, 2005. ACM.

[65] J. Han and M. Kamber. *Data mining: concepts and techniques*. Morgan Kaufmann, 2nd edition, 2006.

[66] Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern Recogn. Lett.*, 24(9-10):1641–1650, June 2003.

[67] M. Heiderich, T. Frosch, and T. Holz. Iceshield: Detection and mitigation of malicious websites with a frozen dom. In *RAID '11*, pages 281–300. Springer, 2011.

[68] M. Hofman. There is some smishing going on in the eu. http://isc.sans.org/diary.html?storyid=6076, March 2009.

[69] R. Hund, M. Hamann, and T. Holz. Towards Next-Generation Botnets. In *Proceedings of the 2008 European Conference on Computer Network Defense*, 2008.

[70] M. Hypponen. Malware Goes Mobile. http://www.cs.virginia.edu/~robins/Malware_Goes_Mobile.pdf.

[71] Internet Identity (IID). Phishing trends report: First quarter 2010. Technical report, 2010.

[72] K. Itabashi. How Trojan.Zbot.B!inf Uses the Crypto API, 2010. Technical Report, Symantec. http://www.symantec.com/connect/blogs/how-trojanzbotbinf-uses-crypto-api.

[73] D. Iyer, A. Mohanpurkar, S. Janardhan, D. Rathod, and A. Sardeshmukh. Credit card fraud detection using Hidden Markov Model. In *Information and Communication Technologies (WICT), 2011 World Congress on*, pages 1062–1066, 2011.

[74] M. Jakobsson and Z. Ramzan. *Crimeware: Understanding New Attacks and Defenses*. Symantec Press Series. Prentice Hall, 2008.

[75] L. Kharouni. Automating Online Banking Fraud. Technical report, Trend Micro Incorporated, 2012.

[76] D. Kirat, G. Vigna, and C. Kruegel. BareBox: efficient malware analysis on bare-metal. In *ACSAC '11: Proceedings of the 27th Annual Computer Security Applications Conference*. ACM Request Permissions, Dec. 2011.

[77] Y. Kou, C.-T. Lu, S. Sirwongwattana, and Y.-P. Huang. Survey of fraud detection techniques. In *Networking, Sensing and Control, 2004 IEEE International Conference on*, volume 2, pages 749–754 Vol.2, 2004.

[78] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage. On the spam campaign trail. LEET'08, 2008.

[79] B. Krishnamurthy and C. E. Wills. On the Leakage of Personally Identifiable Information via Online Social Networks. In *Proceedings of the 2nd ACM Workshop on Online Social Networks*, pages 7–12, New York, NY, USA, 2009. ACM.

[80] B. Krishnamurthy and C. E. Wills. Leakage in Mobile Online Social Networks. In *Proceedings of the 3rd Workshop on Online Social Networks*, June 2010.

[81] T. Lauinger, V. Pankakoski, D. Balzarotti, and E. Kirda. Honeybot, your man in the middle for automated social engineering. In *Proceedings of the 3rd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*, LEET'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association.

[82] M. Lindorfer, A. Di Federico, P. Milani Comparetti, F. Maggi, and S. Zanero. Lines of Malicious Code: Insights Into the Malicious Software Industry. In *Annual Computer Security Applications Conference*, Oct. 2012.

[83] O. B. Longe, V. Mbarika, M. Kourouma, F. Wada, and R. Isabalija. Seeing beyond the surface, understanding and tracking fraudulent cyber activities. *CoRR*, abs/1001.1993, 2010.

[84] F. Maggi. Are the con artists back? a preliminary analysis of modern phone frauds. In *Proc. of the 10th IEEE Intl. Conf. on Computer and Information Technology*, pages 824–831, 2010.

[85] F. Maggi, W. Robertson, C. Kruegel, and G. Vigna. Protecting a Moving Target: Addressing Web Application Concept Drift. In *International Symposium on Recent Advances in Intrusion Detection*. Springer-Verlag, Oct. 2009.

[86] L. Marinos and A. Sfakianakis. ENISA Threat Landscape. Technical report, ENISA, Sept. 2012.

[87] Mashable. Vancouver fans riot as canucks lose stanley cup. http://mashable.com/2011/06/15/vancouver-hockey-riot/.

[88] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems*, 2002.

[89] S. Mhamane and L. Lobo. Internet banking fraud detection using HMM. In *Computing Communication Networking Technologies (ICCCNT), 2012 Third International Conference on*, pages 1–4, 2012.

[90] T. Minegishi and A. Niimi. Detection of Fraud Use of Credit Card by Extended VFDT . In *World Congress on Internet Security (WorldCIS)*, pages 152–159. IEEE, 2011.

[91] T. Moore and R. Clayton. Examining the impact of website take-down on phishing. In *the anti-phishing working groups 2nd annual eCrime researchers summit*, pages 1–13, New York, New York, USA, 2007. ACM Press.

[92] E. W. T. Ngai, Y. Hu, Y. H. Wong, Y. Chen, and X. Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569, 2011.

[93] K. Noto, C. Brodley, and D. Slonim. FRaC: a feature-modeling approach for semi-supervised and unsupervised anomaly detection. *Data Min. Knowl. Discov.*, 25(1):109–133, July 2012.

[94] T. Ormerod. An Analysis of a Botnet Toolkit and a Framework for a Defamation Attack. 2012.

[95] C. Phua, D. Alahakoon, and V. Lee. Minority report in fraud detection: classification of skewed data. *SIGKDD Explor. Newsl.*, 6(1):50–59, June 2004.

[96] C. Phua, V. C. S. Lee, K. Smith-Miles, and R. W. Gayler. A Comprehensive Survey of Data Mining-based Fraud Detection Research. *CoRR*, abs/1009.6119, 2010.

[97] I. Polakis, G. Kontaxis, S. Antonatos, E. Gessiou, T. Petsas, and E. P. Markatos. Using social networks to harvest email addresses. In *Proceedings of the 9th Annual ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 11–20. ACM, 2010.

[98] C. Pollard. Telecom fraud: Telecom fraud: the cost of doing nothing just went up. *Network Security*, 2005(2), Feb. 2005.

[99] M. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 40(3):211–218, 2006.

[100] J. Quah and M. Sriganesh. Real Time Credit Card Fraud Detection using Computational Intelligence. In *Neural Networks, 2007. IJCNN 2007. International Joint Conference on*, pages 863–868, 2007.

[101] Z. Ramzan. Phishing Attacks and Countermeasures. In P. P. Stavroulakis and M. Stamp, editors, *Handbook of Information and Communication Security*, pages 433–448. Springer, 2010.

[102] G. Recourcé. Interpreting contact details out of e-mail signature blocks. In *Proceedings of the 21st international conference companion on WWW*. ACM, 2012.

[103] M. Riccardi, R. Di Pietro, and J. A. Vila. Taming Zeus by leveraging its own crypto internals. In *eCrime Researchers Summit*, 2011.

[104] W. K. Robertson, F. Maggi, C. Kruegel, and G. Vigna. Effective Anomaly Detection with Scarce Training Data. In *NDSS*. The Internet Society, 2010.

[105] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. Dietrich, and H. Bos. P2PWNED: Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In *Proceedings of the 34th IEEE Symposium on Security and Privacy*, San Francisco, CA, USA, May 2013.

[106] C. Rossow and C. J. Dietrich. ProVeX: Detecting Botnets with Encrypted Command and Control Channels. In *Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, July 2013.

[107] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. van Steen. Prudent Practices for Designing Malware Experiments: Status Quo and Outlook. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2012.

[108] K. S. and R. W.V. Online banking fraud detection based on local and global behavior. In *ICDS 2011 : The Fifth International Conference on Digital Society* , pages 166–171, 2011.

[109] D. Sánchez, M. A. V. Miranda, L. Cerda, and J.-M. Serrano. Association rules applied to credit card fraud detection. *Expert Syst. Appl.*, 36(2):3630–3640, 2009.

[110] J. Shawe-Taylor, K. Howker, and P. Burge. Detection of fraud in mobile telecommunications. *Information Security Technical Report*, 4(1), 1999.

[111] K. Sherly and R. Nedunchezhian. BOAT adaptive credit card fraud detection system. In *Computational Intelligence and Computing Research (ICCIC), 2010 IEEE International Conference on*, pages 1–7, 2010.

[112] R. Sherstobitoff. Inside the World of the Citadel Trojan, 2013. Technical Report, McAfee.

[113] N. Soltani, M. Akbari, and M. Javan. A new user-based model for credit card fraud detection based on artificial immune system. In *Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on*, pages 029–033, 2012.

[114] A. K. Sood, R. J. Enbody, and R. Bansal. Dissecting SpyEye – Understanding the design of third generation botnets. *Computer Networks*, Aug. 2012.

[115] F. Stajano and P. Wilson. Understanding scam victims: seven principles for systems security. *Commun. ACM*, 54(3), Mar. 2011.

[116] G. Stringhini, C. Kruegel, and G. Vigna. Detecting spammers on social networks. In *Proc. of the 26th Annual Computer Security Applications Conf.*, ACSAC '10, pages 1–9, New York, NY, USA, 2010. ACM.

[117] M. Syeda, Y.-Q. Zhang, and Y. Pan. Parallel granular neural networks for fast credit card fraud detection. In *Fuzzy Systems, 2002. FUZZ-IEEE'02. Proceedings of the 2002 IEEE International Conference on*, volume 1, pages 572–577, 2002.

[118] D. Tarakanov. Ice IX: Not Cool At All, 2011. Technical Report, Kaspersky Lab. http://www.securelist.com/en/blog/563/Ice_IX_not_cool_at_all.

[119] K. Thomas, C. Grier, J. Ma, V. Paxson, and D. Song. Design and evaluation of a real-time url spam filtering service. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, 2011.

[120] A. van der Merwe, M. Loock, and M. Dabrowski. Characteristics and responsibilities involved in a Phishing attack. In *Proceedings of the 4th international symposium on Information and communication technologies*, WISICT '05, pages 249–254. Trinity College Dublin, 2005.

[121] L. Wang, H. Zhao, G. Dong, and J. Li. On the complexity of finding emerging patterns. *Theoretical Computer Science*, 335(1):15–27, 2005. Pattern Discovery in the Post Genome.

[122] W. Wei, J. Li, L. Cao, Y. Ou, and J. Chen. Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web*, 16(4):449–475, July 2013.

[123] J. Wyke. What is Zeus?, 2011. Technical Report, SophosLabs.

[124] W. Xu and Y. Liu. An Optimized SVM Model for Detection of Fraudulent Online Credit Card Transactions. In *Proceedings of the 2012 International Conference on Management of e-Commerce and e-Government*, ICMECG '12, pages 14–17, Washington, DC, USA, 2012. IEEE Computer Society.

[125] X. Xu, J. Jäger, and H.-P. Kriegel. A Fast Parallel Clustering Algorithm for Large Spatial Databases. *Data Min. Knowl. Discov.*, 3(3):263–290, 1999.

[126] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne. On-Line Unsupervised Outlier Detection Using Finite Mixtures with Discounting Learning Algorithms. *Data Min. Knowl. Discov.*, 8(3):275–300, May 2004.

[127] G. Yan, S. Chen, and S. Eidenbenz. RatBot: Anti-enumeration Peer-to-Peer Botnets. In *Lecture Notes in Computer Science, vol. 7001*, 2011.

[128] T.-F. Yen and M. K. Reiter. Revisiting Botnet Models and Their Implications for Takedown Strategies. In *Proceedings of the 1st Conference on Principles of Security and Trust*, 2012.

[129] Z.-H. Zhou and X.-Y. Liu. Training Cost-Sensitive Neural Networks with Methods Addressing the Class Imbalance Problem. *IEEE Trans. on Knowl. and Data Eng.*, 18(1):63–77, Jan. 2006.