FEEDING PHISHERS

A Thesis

Presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Nicholas James Lynch

July 2009

©2009

Nicholas James Lynch

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Feeding Phishers
AUTHOR: Nicholas James Lynch
DATE SUBMITTED: July 2009

COMMITTEE CHAIR: Dr. Phillip Nico
COMMITTEE MEMBER: Dr. John Bellardo
COMMITTEE MEMBER: Dr. Michael Haungs

Abstract

Feeding Phishers

by

Nicholas James Lynch

Phishing campaigns continue to deceive users into revealing their credentials, despite advancing spam filters, browser and toolbar warnings, and educational efforts. Recently, researchers have begun investigating how fake credentials — or honeytokens — can be used to detect phishing sites and protect users. BogusBiter, one such work, creates sets of honeytokens based on users’ real credentials and sends them alongside real user submissions to phishing sites. In this paper, we present Phish Feeder, an anti-phishing tool which extends the BogusBiter honeytokens generation algorithm in order to create more realistic and authentic-looking credentials. Phish Feeder also employs a “honeytokens repository” which stores generated credentials and provides a lookup service for legitimate sites that encounter invalid credentials. The Phish Feeder client is implemented as a Firefox extension and the repository is implemented as a Java web application. We compare the effectiveness of the Phish Feeder generation algorithm to that of the previous work and find that it is up to four times as effective at hiding real users’ credentials within a set. Furthermore, we find that Phish Feeder introduces only negligible overhead during normal browsing, and a low overhead during credential creation and submission.

Contents

List of Tables	vii
List of Figures	viii
1 Introduction	1
2 Related Work	4
2.1 Phishing Site Detection	4
2.1.1 Blacklists and Whitelists	5
2.1.2 Heuristic-Based Detection	7
2.2 Authentication Mechanisms	10
2.2.1 Browser Modifications	10
2.2.2 New Authentication Paradigms	13
2.3 Fake Credentials	14
3 Design	17
3.1 Generating and Sending Honeytokens	18
3.1.1 Request Monitoring	18
3.1.2 Honeytoken Generation	20
3.1.3 Request Duplication	23
3.2 Honeytoken Repository	24
3.2.1 Site Identification	24
3.2.2 Sending Honeytokens	25
3.2.3 Verifying Honeytokens	25
4 Implementation	27
4.1 Browser Extension	27

4.1.1	Architecture	28
4.1.2	Field Classification	29
4.1.3	Honeytoken Generation	30
4.1.4	Site Identification	33
4.1.5	Uploading to the Honeytoken Repository	34
4.2	Server Web Application	35
4.2.1	Receiving Honeytokens	35
4.2.2	Honeytoken Lookup Service	35
5	Evaluation	39
5.1	Generation Algorithm Effectiveness	39
5.2	Extension Evaluation	43
5.2.1	Correctness: False Positives	44
5.2.2	Correctness: Site Identification	44
5.2.3	Correctness: Field Identification	45
5.2.4	Correctness: Position and Timing Tests	46
5.2.5	Performance Testing	48
6	Conclusions	50
6.1	Future Work	52
	Bibliography	53

List of Tables

3.1	Sensitive Field Classifications	19
3.2	BogusBiter Substitution Example #1	21
3.3	BogusBiter Substitution Example #2	22
4.1	Generation Algorithms Per Field	31
4.2	Example Word Search for <i>TheDuke</i>	32
4.3	Repository Lookup Response Values	37
5.1	Credential Sets for User Survey Question #7	41
5.2	Top 20 English-Speaking Sites	44
5.3	Time Intervals Between Requests – All vs. Real Requests	48

List of Figures

4.1	Honeytoken Repository Submission Format	34
4.2	Honeytoken Repository Lookup Request Format	36
4.3	Honeytoken Repository Lookup Response Format	37
5.1	User Survey Demographics	40
5.2	User Survey Results — Percent Fake Credentials Chosen By Algorithm	42
5.3	Form Input Categorization for Pond Test Sites	46
5.4	Frequency of Real Request Position Within Request Set	47
5.5	Average Request Times to Test Sites With And Without Extension Enabled	49

Chapter 1

Introduction

Phishing is an online form of identity theft in which “phishers” pose as legitimate entities in order to trick users into revealing sensitive personal information. The term comes from the fact that phishers “fish” for credentials from legitimate users. The term phishing was first used circa 1996 to describe scams in which individuals posing as AOL staff tricked users into revealing their billing information [28]. Phishing attacks have continued throughout the years, becoming increasingly prevalent and sophisticated. The Anti-Phishing Working Group (APWG) reported a 20% increase in the number of phishing attacks in the second half of 2008 [20]. Originally, phishing sites were run by individual phishers working independently; today, phishing campaigns are carried out *en masse* under the supervision of organized crime [15].

For each phishing campaign, phishers choose an established legitimate site to mimic. Phishing sites are then designed which imitate the look and feel of the legitimate site and emails linking to these sites are sent to potential victims. These emails typically include some “call to action”. For example, a phishing email claiming to be from a financial institution may ask that users follow a

link and confirm their account details to verify their identity or prevent account deletion. Rather than linking to the financial institution itself, the email will link to a phishing site disguised as the financial institution.

Phishers most frequently choose to pose as financial institutions and online payment services, although any popular site may be chosen, as usernames and passwords are often re-used across multiple sites. Phishers usually send emails to a large number of recipients and impersonate a popular site to increase the odds that a recipient will have an account at that site and that the email will appear genuine. The same legitimate sites are chosen frequently; in January of 2008, 80% of the phishing campaigns reported to APWG represented only 15 of the 131 reported “hijacked brands” [19]. In contrast, “spearphishing” is a technique in which emails are sent to smaller, more targeted groups. This technique allows phishers to run more tailored campaigns, and potentially fool a larger percentage of their victims.

Researchers have responded to the threat of phishing by creating a variety of anti-phishing solutions, designed to detect phishing sites and warn users of potential danger. However, studies have shown that users may frequently ignore these warnings, especially if a detection scheme is prone to false positives [32]. Much anti-phishing research concentrates on preventing users from disclosing their credentials to phishers, either by discouraging them from submitting sensitive data on untrusted sites or by warning them when a site appears malicious. While this is optimal, some users may still ignore warnings or unknowingly submit their credentials to phishing sites. Protecting users in this case is a task not fully explored in the current literature.

Recent research has examined how fake credentials can be used to protect users, even when users are tricked into revealing their credentials. In this paper,

we introduce Phish Feeder, an anti-phishing tool which expands on this research. Phish Feeder uses intelligently-generated fake credentials to clutter and devalue phishing databases and protect end users. Every submission on a phishing site triggers the generation of fake credentials, which are sent alongside the real credentials to the phishing site. The goal is to prevent phishers from deterministically knowing which credentials are valuable, forcing them to either guess or test each credential individually. Additionally, a Phish Feeder central repository provides a lookup service which can be used by legitimate sites; this service can determine if invalid credentials originated from a Phish Feeder client, which may be used to identify phishers who use these credentials.

The rest of this document is structured as follows. Section 2 describes the previous work done in this area. Sections 3 and 4 discuss the high- and low-level details of the system and its design. Section 5 covers the evaluation and subsequent analysis of the system. Section 6 summarizes our conclusions and presents ideas for future work.

Chapter 2

Related Work

As phishing attacks grow more sophisticated, so do the techniques used to detect and defeat them. The Anti-Phishing Working Group (APWG) currently receives reports of over 20,000 new phishing sites every month [1]. Researchers have responded by creating and improving a wide array of techniques to prevent phishers from succeeding, including phishing site detection, new paradigms for authentication, and fake credential generation.

2.1 Phishing Site Detection

A phishing site's success depends on how well it can mimic its target; pages that bear no resemblance to the target site or contain obvious differences are much less likely to attract users' credentials. Thus, phishers focus on making their site as close to an exact duplicate of their target as possible, often copying the pages directly and changing only a few fields. Several studies have shown that users have a hard time distinguishing real sites from their phishy counterparts [14] [21]. Due to this, a significant amount of research has been devoted to detection

of phishing sites and corresponding user notification schemes.

The techniques proposed range from leveraging lists of known safe and unsafe sites to static and dynamic analysis of various page properties. Many of these techniques have been implemented as browser extensions and several have already been incorporated directly into the major web browsers.

2.1.1 Blacklists and Whitelists

In the context of phishing site detection, a blacklist is a list of known phishing sites. Many phishing detection tools — including those built into major web browsers — use blacklists to some extent. A recent study found that eight of ten popular anti-phishing toolbars used blacklists as all or part of their detection scheme [11]. However, blacklists are not an all-inclusive solution to phishing, as sites are only added to blacklists after they have been discovered and reported. Blacklists may be updated by a variety of sources, from spam filters to individual user submissions; depending on the trustworthiness of these sources, these new entries may need additional verification before they are added to the blacklist [31]. Users who visit phishing sites before they have been added to the blacklist will receive no warning; for this reason, blacklists are often frequently updated - the Firefox blacklist is updated “every 30 minutes or so” [25].

In contrast to blacklists, whitelists are lists of known safe or trusted sites. Whitelists are generally pre-populated with popular, trusted sites and are updated by end users as needed. Whitelists are infrequently used as a complete anti-phishing solution; instead, whitelists are usually checked before running some other detection algorithm to reduce false positives.

In 2005, Kirda and Krugel proposed AntiPhish, a browser extension which

protects users from submitting their credentials to unauthorized sites by maintaining a whitelist of known sites [22]. As users browse and enter their credentials, they can choose to store these credentials and the associated site’s domain in AntiPhish’s whitelist. Each time a user submits their data to a website, AntiPhish checks the whitelist; if the user enters known credentials into an unknown or untrusted site, AntiPhish displays a warning and allows them to cancel the request.

Cao et al. introduced the Automated Individual White-List (AIWL), an anti-phishing tool based on an individual user’s whitelist of known trusted sites [6]. AIWL is trained to recognize successful and unsuccessful login attempts through the use of a Naive Bayesian classifier. After repeated successful login attempts, AIWL prompts the user to add a site to the whitelist. The whitelist is composed of each trusted site’s Login User Interface (LUI) — the site’s URL, a hash of its security certificate, a hash of the credential for that site, valid IPs for the URL, and the HTML DOM path to the username and password input elements. Users are warned when they submit their credentials to a site which is not present in the whitelist. This system assumes that users will only repeatedly submit credentials to legitimate sites, and that all other sites are potentially malicious.

Several other phishing detection tools maintain whitelists to allow users to store exceptions and prevent repeated false positives in detection [17] [29]. Limiting repeated false positives via whitelists may improve the user experience, but it introduces potential risks if a phishing site is accidentally or maliciously added to the whitelist or if a previously benevolent site becomes malicious.

2.1.2 Heuristic-Based Detection

Blacklists and whitelists can only be used to recall previously identified sites. In order to automatically detect phishing sites, a variety of heuristics have been developed. The tools in this section use one or several heuristics to identify phishing sites based on common patterns and characteristics. All of these tools share common ground in that the detection is not foolproof: as the heuristics are not absolute, they are prone to both false positives and false negatives.

In 2004, researchers from Stanford proposed an Internet Explorer toolbar they dubbed SpoofGuard [10] to detect malicious sites. SpoofGuard uses a configurable, weighted algorithm to determine the probability that a page is malicious based on a variety of heuristics. SpoofGuard analyzes URLs for patterns used by phishing sites, compares images on the site to those from popular domains, and hashes and compares post data to stored sensitive data. SpoofGuard also allows the user to configure the algorithmic weights in order to control the level of false positives. Subsequent studies have found SpoofGuard to have a 90% true positive rate, but a 35–48% false positive rate [24] [37].

Chen and Guo [9] improved on the URL analysis work done in SpoofGuard. After studying 203 phishing emails collected from the APWG [1], they identified five categories of website link obfuscation employed by phishing sites and measured their frequency. Links are classified as belonging to a phishing site if one of the following is true:

- The anchor link and text use DNS names, but the DNS names do not match each other
- The anchor link uses a dotted decimal IP address instead of a DNS name

- The anchor link is encoded or uses a username/password prefix
- The anchor link uses a DNS name but the anchor text does not
- The anchor uses a redirect vulnerability from a legitimate site

The authors obtained a 96% success rate identifying phishing links from the 203 phishing emails studied; however, no analysis was performed of the tool’s false positive rate.

Around the same time, Garera et al. [17] also developed techniques for identifying phishing sites based on properties of the site’s URL, including if the hostname is an IP address or if the hostname or path contains the domain being targeted. Additionally, Garera et al. proposed several new heuristics: the site’s Google PageRank, presence in a pre-defined whitelist, and certain keywords within the URL. The authors acquired a base set of 2,508 URLs — about half of which were known phishing sites. They trained their tool on the first 66% of the sites (chosen at random), then ran their tool against the remaining URLs. They found their tool had a 88% true positive rate, and a 0.7% false positive rate.

Likarish et al. from the University of Iowa proposed B-APT, a means of identifying phishing sites by training and utilizing a Bayesian filter [24]; Bayesian filters are already widely used to identify spam emails, with extremely positive results. If a website is not in B-APT’s whitelist, it is tokenized and analyzed by the Bayesian filter, which returns a probability that the site is a phishing site. Likarish et al. trained B-APT with legitimate sites from the Alexa Top 500 [5] and phishing sites from Broadband Report’s Phishtrack [13]. When tested against 60 phishing and 60 legitimate sites, B-APT obtained a 100% true positive and 3% false positive rate of detection.

Along a similar vein, Zhang et al. developed CANTINA [37], a tool for detecting phishing sites based on a generated lexical signature of a page. The authors used a TF-IDF (term frequency — inverse document frequency) algorithm, which calculates how “important” a word is in a document by measuring its frequency in the document and its frequency across all documents in a set. The five terms on the page with the highest TF-IDF score are fed to Google; if the current page does not appear within the top 30 results, then it is determined to be a phishing site. The authors also used a variety of heuristics similar to those already presented to lower their false positive rate. With these heuristics, the tool achieved an 89% true positive and a 1–3% false positive rate.

In 2008, Ronda et al. created iTrustPage [29], in an attempt to reduce the false positives and false negatives associated with the heuristics of the above-mentioned papers. iTrustPage is not a fully automated tool and partially relies on user input to make the “correct” decision about the validity of a site. Like CANTINA, iTrustPage performs a Google search based on key terms for the page; unlike CANTINA, the search terms are supplied by the user. The first time users encounter a form not present in iTrustPage’s pre-defined whitelist, they are prompted for a set of search terms, which are used to make a Google search. If the current site is not present in the results, the top sites from the results are displayed as an overlay on top of the current page, and the user is given the option to navigate to any of these sites. Ronda et al. deployed iTrustPage and collected statistics on how many pages were visited and how frequently the tool required user intervention. They found that iTrustPage disrupts the user on less than 2% of pages visited after a week of use; of that 2%, users chose to bypass iTrustPage’s validation 88% of the time. Due to privacy concerns, the authors did not gather browsing history and cannot determine if iTrustPage prevented

any users from becoming victims of phishing attacks.

2.2 Authentication Mechanisms

Although many tools have been developed to detect phishing sites and alert users, phishing continues to be a serious problem. One study suggests that this is because current user notification techniques are inadequate; Wu et al. found that up to 60% of users were fooled by a phishing site, even when warned by an anti-phishing toolbar. To combat this, researchers have studied different authentication mechanisms to help users enter their credentials only at valid sites. These new mechanisms include authentication protocols, visual feedback techniques, password hashing, and three-factor authentication schemes.

2.2.1 Browser Modifications

The following techniques combine some new authentication mechanism with a new visual element to help users identify when they are at a trusted site. Most of these proposals suffer from a common shortcoming — each of these tools must be set up on a per-computer or per-browser basis, resulting in a negative user experience for those who interact frequently with multiple workstations.

Dhamija and Tygar [12] suggested pairing a new authentication strategy with a visual identification system they labeled Dynamic Security Skins. The browser is modified to include a separate “trusted password window”, which is solely used for password entry and to display security information. The trusted password window uses a user-chosen image as a background image, in order to greatly reduce the possibility of the window being spoofed by a phisher. Users input

their username and password into the trusted password window to login to a web site. A hash is exchanged between the user and the web server, which is then used to create an image via a visual hash algorithm. The hash is exchanged via an adaptation of the Secure Remote Password protocol (SRP). This protocol requires that users prove they know their password and the server knows the user’s verifier (a hash of the password and a salt), without either party directly exchanging this information; see [34] for more information about the SRP protocol. Phishing sites, which presumably do not have the user’s verifier, cannot properly authenticate via the SRP protocol and no hash image will be generated. If the protocol succeeds, the trusted password window and web page both display the hash image and users can compare the images to ensure they are interacting with a trusted site.

Wu et al. also designed a browser sidebar specifically for handling user logins, named WebWallet [33]. WebWallet forces the user to login through WebWallet by detecting and disabling all password fields on a page. When users submit credentials to an untrusted site for the first time, they are asked to choose which site they intended to submit to from a list. If the user choice’s does not correspond with the current site, a link is provided to the intended site. Users may also view security details about the current site, including presence of a verified certificate, the length of time the domain has been registered, and the site’s country of origin. Users may also store their credentials (as a domain, user, and password tuple) for future logins. Forms for which there are saved credentials are automatically populated on recurring visits when the user hits the predefined “security key”. The authors argue that this saved credential interface provides protections and simultaneously simplifies repeat logins to trusted sites.

PwdHash, presented by Ross et al., uses a password hashing algorithm in order to secure user’s passwords [30]. Any time a user needs to enter their pass-

word, they first enter the PwdHash-defined password prefix (“@@”). As they type their password, each character is saved and replaced with a fixed sequence to thwart JavaScript-based password-stealing attacks. On form submission, the password sequence is replaced with a hash of the user’s password, salted with the current domain. PwdHash requires the browser extension to be installed on each machine the user needs to login at; they mitigate this weakness by providing a site which mimics PwdHash’s capabilities — users can enter a site domain and their password and receive the hashed password for that domain. This calculation is done entirely via client-side JavaScript to avoid transmitting the user’s password over the network.

In 2006, Yee and Sitaker [35] also presented a password hashing-based solution to phishing. With PassPet, the user registers a unique username at the PassPet server via a browser extension. The user is assigned a random animal icon and “pet name”, to be displayed in the PassPet toolbar; like the personal image background in the Dynamic Security Skins paper, these personalizations reduce the possibility of spoofing the browser interface. During normal use, users assign a unique label to each site they wish to login to; users are notified when they attempt to use the same label for multiple sites, which may indicate phishing attacks. PassPet uses a user-supplied “master secret”, the user’s PassPet username, and the site label as inputs to a password hashing algorithm, which is used to generate a unique password for each site the user authenticates with. As long as labels are unique across sites, the passwords supplied to each site will be different. Users do not know their individual passwords, so they cannot be tricked into providing them to a phishing site. This means that like PwdHash above, users cannot log into their accounts without installing the tool on each workstation they frequent.

2.2.2 New Authentication Paradigms

The above authentication techniques all require some modification to the existing browser interface. While changing existing interfaces is possible, it requires developer effort, and must be done separately for each browser. Alternative authentication techniques have been proposed which allow users to login to sites without browser modifications; some merely require server-side changes, while others also require a third, trusted device to ensure secure authentication.

Rather than create new browser interfaces, Adida [4] suggests a two-factor web authentication scheme based on browser bookmarks called BeamAuth. Using BeamAuth, users are given a custom bookmark, which they use to login to a site; during first login, this bookmark is provided to the user through a second-channel authentication mechanism, such as an SMS or email verification. The bookmark's URL fragment contains a user-specific secret token. When users navigate to the site via the bookmark, JavaScript on the page reads in the secret token, stores it as a local variable, populates the username in the login form, and removes the fragment from the address bar. When users submit the form with their password, the JavaScript intercepts the call and sends a secure hash of the user's token and password to the server. This technique means that the website always requires both the token and password to log in and the server only stores a hash of the token and password. Additionally, users who are tricked into revealing their passwords at a phishing site have released only one of the two forms of authentication necessary for the site.

Parno et al. suggest Phoolproof: an authentication mechanism which uses an external trusted device, such as a cellphone, to protect users from phishing attacks [26]. The device acts as a third-party oracle during both initial account

setup and subsequent logins, providing information to the user’s browser and verifying stored server certificates. During an account setup, the device generates a public/private key pair and stores this pair with the web server’s security certificate. The public key is then communicated via the user’s browser and associated with the user’s account on the server. To later establish a secure connection with the web server, the server’s stored certificate is first verified by the device, then the shared public key is used to establish a SSL/TLS connection. Once a secure connection is established with the assistance of the device, the user is free to browse the site normally. The advantage of this mechanism is that a phisher must compromise both the user’s password and device in order to gain access to the account.

2.3 Fake Credentials

All of the above areas of research take a defensive stand to identify phishing sites or protect a user’s credentials. Recently, research has been conducted studying potential offensive measures against phishing sites, particularly through the use of falsified data.

In 2006, Madhusudhanan et al. proposed an offensive detection tool named PHONEY designed to function between a mail server and mail client [8]. PHONEY analyzes incoming messages based on a set of heuristics. If the message or any sites it links to contain a form requesting sensitive information, the message is tagged as malicious, and the tool proceeds with its detection algorithm. The fields of the form are then populated with fake data from PHONEY’s database based on each field name. The form is then submitted to the site, and the response is analyzed by a rule-based decision engine to determine if the site represents

a phishing attack. Based on their sample set of 20 phishing emails and some legitimate emails, the authors claim PHONEY is able to detect all email-based phishing attacks without false alarms.

Yue and Wang later proposed BogusBiter, a browser extension which feeds a large number of fake credentials to a phishing site [36]. BogusBiter’s design has two main goals — to fill the phishers’ databases with fake credentials and allow legitimate sites to detect when stolen credentials are being used. When users visit a phishing site and submit their credentials, BogusBiter intercepts the request and creates a set of fake credentials based on the user’s credentials. All of the credentials are sent to the phishing site simultaneously. The fake credentials are created using numerical and alphabetical substitution; the goal of the substitution strategy is to create fake credentials such that picking out the real credentials from the set is extremely difficult. Additionally, the authors propose a mechanism which can detect phishers who use these fake credentials. The properties of the algorithm ensure that rerunning the algorithm with any of the fake credentials as the input will result in a new set of generated credentials which contain the real credentials. A legitimate site may use this to its advantage by running this algorithm on invalid credentials it receives; if any legitimate credentials are present in the generated set, then the invalid credentials were likely generated by BogusBiter, and the user providing the credentials is likely a phisher. The site can then take appropriate actions to block the phisher, warn the user, and/or block the user’s account temporarily.

In this paper we introduce Phish Feeder, an anti-phishing tool which uses carefully crafted fake credentials in order to protect users and devalue phishing databases. Phish Feeder was largely inspired by the BogusBiter tool, and shares some of the same strategies of request creation and credential generation. Like

BogusBiter, our tool also feeds a large number of fake credentials to suspected phishing sites; however, Phish Feeder builds on and extends BogusBiter in the following key ways:

- In addition to handling usernames and passwords, Phish Feeder also supports other types of sensitive information, including credit card and social security numbers.
- Phish Feeder extends the substitution algorithm presented in BogusBiter to better obfuscate the legitimate credentials within the credential set.
- Rather than creating a reversible substitution algorithm, Phish Feeder employs a secondary “honeypot repository” as an interface for determining if invalid credentials were generated for a phishing site.

Chapter 3

Design

Phish Feeder is composed of two distinct parts: a client-side browser extension to create and send fake data and a central server to act as a repository for this data.

The browser extension protects users by accompanying any submissions to a phishing site with a set of fake submissions. Rather than receiving one valuable request, the phisher receives a set of requests, only one of which contains real credentials. Phishers have the option of attempting to manually identify the real credentials or accepting all of the credentials and polluting their database. The former requires extra effort on the phisher's part, while the latter may reduce the value of the database.

Phish Feeder also provides a way of identifying those phishers who choose to keep and later use the fake credentials. All of the generated credentials are stored in a central honeypot repository, which provides a lookup service. Legitimate sites which receive invalid credentials may query the repository to determine if these credentials were generated by Phish Feeder; if the credentials are present in

the repository, there is a high probability that the user providing these credentials is a phisher.

3.1 Generating and Sending Honeytokens

The Phish Feeder browser extension is installed on each end-user’s computer. The extension is responsible for detecting form submissions on a phishing site, analyzing the forms, generating fake credentials, and sending the fake credentials to the phishing site alongside the real request. The phishing site receives all of the credentials and cannot easily determine which credentials are legitimate, even with manual inspection. The extension also submits the generated credentials to the honeytoken repository, where they are stored for future lookup requests.

3.1.1 Request Monitoring

Once installed, the Phish Feeder extension does not collect information or generate honeytokens until it detects the user is visiting a phishing site. Phish Feeder currently uses Firefox’s built-in phishing detection to determine when the user visits a phishing site, although additional heuristics could be included in the future. Once a phishing site is reached, Phish Feeder begins monitoring form submissions. Phish Feeder ignores all form submissions on presumed legitimate websites — no information about these forms is collected or stored.

As a form on a phishing site is submitted, information about the form is collected and analyzed. Phish Feeder stores information about the form itself (e.g. action, method) as well as information about each input field on the form (e.g. type, value). This information is used to generate the honeytokens and

Name	Determined By		
	Type	Location	Value
Password	✓		
Username	✓	✓	
Email			✓
Social Security Number			✓
Credit Card Number			✓

Table 3.1: Sensitive Field Classifications

duplicate the original request (see Sections 3.1.2 & 3.1.3 for further details).

Field Classification

In order to produce the most realistic credentials, Phish Feeder’s honeytoken generation algorithm provides multiple different generation strategies. These generation strategies are applied based on Phish Feeder’s classification of each field. As the form is submitted, Phish Feeder gathers information about each field on the form and uses this information to classify specific fields as containing potentially sensitive information. Honeytoken values are only generated for fields which contain sensitive information; this reduces the size of the honeytoken set, and allows Phish Feeder to make more intelligent honeytoken generation decisions. After all sensitive fields have been classified, the appropriate generation strategies are applied to each field to create the honeytoken values.

Phish Feeder initially considers all input elements of the form and then identifies potentially sensitive fields. Table 3.1 provides a list of the field classifications. Fields can be classified based on a combination of field type, location within the form, or the field’s value. Previous work in this area relied on manual user identification [29] or only considered the password fields [36] of a form. Most of the field classifications apply to a single field, although some classifications can apply

to multiple fields within a form (e.g. a credit card number broken up into four separate fields).

3.1.2 Honeytoken Generation

Phish Feeder uses the information gathered about the form to generate a set of N honeytokens for the request. Each honeytoken contains a value for every field on the form. For fields identified as sensitive, a new value based on the original is generated and used; for all other fields, the original value is used. The honeytoken generation algorithm used is based on that of BogusBiter [36]; what follows is an overview of the BogusBiter algorithm, followed by a description of how the algorithm has been extended for this work.

BogusBiter algorithm

The BogusBiter generation algorithm creates honeytokens for usernames and passwords; for simplicity, only usernames will be referenced here, as both usernames and passwords are treated identically by the algorithm. The algorithm works as follows: first, a character of the username is identified as the replacement character (rc). The replacement character is defined as the first digit in the username, or the first letter if no digit exists. A position i is randomly chosen between 1 and N ; this position represents the location of the real username within the generated set of N usernames. For each position j , the algorithm replaces rc with a digit (or letter) $j - i$ positions up in the sequence “0123456789” (or the English alphabet). The sequence is wrapped if necessary (i.e. ‘a’ follows ‘z’, ‘9’ precedes ‘0’). Table 3.2 demonstrates the effect of the BogusBiter algorithm on the username/password pair *mcsmith/Fuzzycat15*, with $N = 4$, and $i = 3$.

Position	Username	Password
$j = 1$	kcsmith	Fuzzycat99
$j = 2$	lcsmith	Fuzzycat09
$\rightarrow i = 3$	mcsmith	Fuzzycat19
$j = 4$	ncsmith	Fuzzycat29

Table 3.2: BogusBiter Substitution Example #1

Legitimate sites may use the BogusBiter algorithm to detect if invalid credentials were generated by BogusBiter. Due to the linear and deterministic nature of the algorithm, re-running the algorithm with a size of $2(N - 1)$ and using any generated username/password pair as the input will create a new credential set containing the original (real) username/password pair. Legitimate sites may provide failed login credentials to this algorithm and search the resultant set for legitimate username/password pairs. If found, this would suggest — with high probability — that the failed credentials were generated by BogusBiter.

It is possible that the server-side algorithm may produce collisions when used on a single field, reporting that credentials were generated by BogusBiter when they were not. This is especially true when digit-based substitution is used (e.g. both *surfdude1* and *surfdude2* may be valid usernames). However, the odds of correctly generating a valid username and password pair is equivalent to randomly guessing user passwords [36], so collisions are much less likely when username and password are considered together.

The BogusBiter generation algorithm works adequately for most usernames and passwords containing digits, and those conforming to specific patterns (e.g. a username scheme of *[first initial][last name]*). However, it does a poor job of concealing the real credentials when they do not contain digits, and do contain dictionary words, names, or other patterns. Table 3.3 shows the result of the

Position	Username	Password
$j = 1$	kcsmith	Duzzycat
$j = 2$	lcsmith	Euzzycat
$\rightarrow i = 3$	mcsmith	Fuzzycat
$j = 4$	ncsmith	Guzzycat

Table 3.3: BogusBiter Substitution Example #2

BogusBiter algorithm on the username/password pair *mcsmith/Fuzzycat*. In this example, digit replacement is not used, and the real credential set is much more recognizable to a human.

Phish Feeder Algorithm

The Phish Feeder algorithm extends the BogusBiter algorithm to provide a more robust generation process which can adequately handle a wider variety of inputs. Like the BogusBiter algorithm, Phish Feeder determines one or more characters in the value to be replaced. This replacement string is chosen from one of three different replacement strategies based on the field classification. The replacement strategies are attempted in the following order:

- **Digit Replacement:** If the value contains one or more digits, the first group of digits is chosen.
- **Word Replacement:** If the value contains a dictionary word or popular name, that word or name is chosen.
- **Alpha Replacement:** If no digit or word is present, the first letter is chosen.

The principal new functionality of the generation algorithm is the addition of word- and name-based replacement. Many usernames and passwords may contain

dictionary words (or names), but not digits. The BogusBiter algorithm would change only the first letter of the value, resulting in potentially obvious fake credentials when compared to the originals. By replacing whole words instead of single characters, the likelihood of detection is reduced.

Phish Feeder also uses the field’s classification to aid in the generation of the honeypoken values. For instance, for a field classified as “email”, only the username (everything left of the ‘@’ sign) is considered for replacement; another example is credit card fields, in which the digits are crafted to pass the credit card Luhn validity check.

Details on the implementation of the honeypoken generation algorithm can be found in Section 4.1.3. A user survey was conducted to determine the effectiveness of the improvements; see Section 5.1 for further information.

3.1.3 Request Duplication

After the necessary information has been collected about the form and the honeypokens have been generated, N requests are created. Each of these requests is carefully crafted to be a duplicate of the original, with the exception of the field values; the request method, origin, referrer, and all other applicable headers are copied exactly. After the requests are created, they are sent to the server alongside the original request. The server receives all $N + 1$ requests sequentially. The position of the real request within the set is randomized to prevent phishers from distinguishing the original request from the others in a deterministic manner.

3.2 Honeytoken Repository

The BogusBiter algorithm is partially deterministic and reversible; this strategy allows legitimate servers to create new credential sets using any invalid credentials as input. If any legitimate credentials are present in the newly generated set, the invalid credentials were most likely created by BogusBiter. Unfortunately, this reversibility comes at a price — the algorithm is limited to a deterministic formula and works well on certain input sets, but quite poorly on others.

The Phish Feeder algorithm opts for more versatile honeytoken generation, but this comes at the loss of reversibility. Instead, Phish Feeder utilizes a central repository to store the generated honeytoken values (see Section 4.2). After each request, the set of generated honeytokens is collected and sent to a central server, along with the identity of the targeted site, if possible. This repository stores all of the credentials provided and allows legitimate sites to query it to determine if invalid credentials were generated by Phish Feeder.

There is a possibility that legitimate sites will be provided with invalid credentials — due to user error or other innocent means — that are coincidentally present in the repository. We refer to this as a collision, and we attempt to limit the effects of collisions by returning detailed responses to lookup queries (see Section 4.2.2) and associating honeytokens with their intended sites whenever possible.

3.2.1 Site Identification

Each phishing site deliberately mimics a specific site’s interface and design. In order to increase the value of the honeytoken repository, honeytokens are

associated with this mimicked site if at all possible. This allows the repository lookup service to determine if credentials were intended for a specific legitimate site, which provides more information to the legitimate sites and reduces the possibility of a repository collision.

Some previous works use a form of target site identification as part or all of their phishing site detection mechanisms; current techniques include url analysis, image recognition, link analysis, and site structure analysis [10] [16]. Phish Feeder employs a simple but effective heuristic for determining a phishing site’s target. The domain of each link on the page is gathered and tabulated; the domain with the highest link count is assumed to be the target site.

3.2.2 Sending Honeytokens

The honeytoken repository acts as a central server for all generated honeytokens. To accomplish this, each Phish Feeder extension sends information about each set of honeytokens it generates to the repository. For each sensitive field on the form, Phish Feeder sends the field classification and hashes of all of the generated values for that field. Phish Feeder also sends the domain of the targeted site, as determined by the site identification heuristic. The repository stores this information for use by the lookup service.

3.2.3 Verifying Honeytokens

Target sites can query the repository to identify credentials as Phish Feeder-generated honeytokens. When a legitimate site receives invalid credentials, it queries the honeytoken repository lookup service with these values and types. The repository compares the received credentials to those present in the database

and returns one or more statuses to the legitimate site, depending on the accuracy of the match. Further details are discussed in [Section 4.2.2](#).

In this chapter, we have discussed the high-level design of Phish Feeder; the next chapter expands on some of these ideas and provides a more in-depth discussion of the algorithms and policies of the tool.

Chapter 4

Implementation

Phish Feeder is implemented as two distinct components: a browser extension which generates and sends honeytokens, and a server application which stores the honeytokens and allows querying from legitimate websites. In this chapter, we discuss the Phish Feeder design and algorithms at a lower level of detail, to facilitate further understanding and reproducibility.

4.1 Browser Extension

Browser extensions allow developers to add new functionality to popular web browsers. End-users may add these extensions to their browser in order to change the way a page is rendered, prevent content from being included, aid web development, or perform any other variety of tasks. Phish Feeder's browser extension is responsible for collecting form information, generating honeytokens, and creating and sending requests.

Phish Feeder is implemented as an extension to the popular Firefox web

browser. The extension is primarily implemented in just under 2,500 lines of C++. Although most common Firefox extensions can be written using only a combination of XUL and JavaScript, we chose C++ as our primary development language. Using C++ provided access to certain Firefox interfaces which are not available from JavaScript, and gave us the possibility of linking with additional third-party libraries if necessary. Additionally, C++ extensions may achieve better performance than their JavaScript counterparts [18].

4.1.1 Architecture

While the vast majority of the extension is implemented in C++, other languages and configuration files are necessary in order to integrate the extension with Firefox. A Firefox extension uses the XML User Interface Language (XUL) to define all user interface components; the XUL files can also include JavaScript to control extension behavior. As Phish Feeder has no user interface or notifications, the XUL file contains only basic JavaScript which loads the C++ component.

In order to interact with the C++ code from JavaScript, an XPCOM interface was defined. XPCOM (cross-platform component object model) is a framework for writing cross-platform, cross-language software, similar in nature to the Microsoft COM. Interfaces are defined using an Interface Description Language, and can be implemented in JavaScript, Java, Python, or C++ [7]. Phish Feeder defines one simple interface — `nsIFeeder` — containing two methods: `register()` and `unregister()`. When Firefox is opened, the call to `register()` loads and initializes Phish Feeder; on close, the `unregister()` call performs clean-up operations. All necessary browser interactions and monitoring are performed at the

C++ layer.

4.1.2 Field Classification

Phish Feeder collects information about each form on a suspected phishing site as it is submitted; this information is used to classify the form's inputs. Fields may be classified by submitted value, type, or type and field position. Each field receives at most one classification, and the classifications are determined in the following order:

1. **Email:** Input's value matches the pattern of an email address.
2. **Credit Card Number:** Input's value is 13 to 16 digits, and passes the Luhn validity check.
3. **Social Security Number:** Input's value is 9 digits, optionally separated by spaces or dashes into the form (`###-##-####`).
4. **Password:** Input's type is "password"; its value does not match any of the above criteria.
5. **Username:** Input's type is "text", and it is either one field before or after a password field; its value does not match any of the above criteria.
6. **Other:** Input's type is "text", contains at least one digit, and does not meet any other criteria.

Additionally, the Credit Card Number and Social Security Number classifications may represent multiple fields on a form. The following rules determine if a group of fields is classified as either of these types.

1. **Credit Card Number:** Four consecutive inputs contain four digits each; the aggregate value passes the Luhn validity check.
2. **Social Security Number:** Three consecutive inputs contain three, two, and four digits, respectively.

4.1.3 Honeytoken Generation

Phish Feeder generates a set of values for every classified field. A set of honeytokens is then created, each of which contains one of the generated values for each classified field and the original values for all other fields. As the user submits to a phishing site, a set of requests is generated and sent alongside the original request. Each of these created requests contains the values from one of the generated honeytokens.

For each classified field, a set of N values is generated. N is initially set to 9, although this is configurable. This number was chosen based on previous work, which found that more than 10 subsequent requests was perceived as a malicious attack on some legitimate sites.

There are four distinct generation algorithms used to create the honeytoken values. Each algorithm has criteria which must be satisfied in order for the algorithm to be used; if the algorithm criteria are not met, the next algorithm is tried. Table 4.1 shows the algorithm to field correlation. The algorithms are attempted in the following order: Card Number, Digit, Word, and Character. This order puts the most successful algorithms — those that have the highest chance of obfuscating the real credentials — first. What follows is a description of each algorithm:

Name	Card Number	Digit	Word	Character
Username		✓	✓	✓
Password		✓	✓	✓
Email		✓	✓	✓
Credit Card Number	✓			
Social Security Number		✓		
Other		✓		

Table 4.1: Generation Algorithms Per Field

Card Number:

The Card Number generation algorithm is only used for credit card number fields. Credit card numbers generally have a issuer-specific prefix of 4-6 digits, followed by an account number, and the Luhn check digit. The Card Number generation algorithm generates a random valid credit card number for each field value. To do this, it keeps the first 6 digits, randomly generates a new account number, and sets the check digit to 0. The Luhn-algorithmic sum is then calculated, and the check digit is adjusted as necessary in order make the Luhn sum evenly divisible by 10; this indicates a syntactically valid credit card number.

This algorithm generates valid credit card numbers only in the sense that they will pass the Luhn validity check. We are not providing any new or novel information to the phishers via this generation strategy, as valid credit card numbers can be easily generated using this or similar algorithms.

Digit:

The Digit generation algorithm is used for all other fields which contain at least one digit. The last digit in the first group of digits is chosen as the replacement digit. The replacement digit is then replaced with a different random digit

N times. Each result is guaranteed distinct with a value of $N \leq 9$; if N is larger, repetition will occur.

Word:

If no digits are present in the field value, the Word generation algorithm is tried next. The Word generation algorithm identifies the replacement word as the largest dictionary word or name present in the field value; if the field value itself is a word or name, the entire field value is chosen as the replacement word. Otherwise, substrings of the word are identified and tested for dictionary presence in descending order of length. For example, for a field value of length M , the field value is first tested. Then the two substrings of length $M - 1$, starting at indexes 0 and 1, are tested. The minimum word length for consideration is four characters.

Word Length	Substrings
7	TheDuke
6	TheDuk, heDuke
5	TheDu, heDuk, eDuke
4	TheD, heDu, eDuk, <u>Duke</u>

Table 4.2: Example Word Search for *TheDuke*

Table 4.2 shows the substrings evaluated for the username *TheDuke*. In this example, *Duke* is chosen as the replacement word. If a dictionary word had been found before *Duke*, it would be chosen instead, as longer words are preferred.

Once the replacement word is identified, it is replaced with random words of the same length N different times to produce the N values for the field. The replacement words are chosen non-deterministically so as not to reveal any information about the original value. Currently, the built-in Firefox dictionary is

used for word lookup and replacement, although Aspell [2] or any other dictionary library could be used in the future.

Character:

If the criteria for the other generation algorithms are not met, the Character generation algorithm is used. The first alphabetic character is chosen as the replacement character, and is replaced with a different random character N times. Case sensitivity is retained; the replacement character is replaced with a letter of the same case to preserve capitalization and patterns in the field value.

4.1.4 Site Identification

Phish Feeder uses a link-based site identification heuristic in order to determine the targeted site for a phishing attack. This information is used later by the honeypot repository to reduce the chance of collisions.

Phishing sites are often almost direct copies of legitimate sites; phishers change the site as little as possible to preserve the image that the site is legitimate. Additionally, rather than host the entire legitimate site, only one or two pages is copied to reduce overhead. Due to these factors, phishing pages often contain links back to the original legitimate site. By examining the phishing site for these links, we may be able to identify the legitimate site.

As the form on the phishing site is submitted, the domain of each link on the page is observed and counted. As the goal of the heuristic is to determine the identity of an external site, only absolute links to external domains are tabulated; relative links, fragments, and JavaScript-based links are not added to the count. The external domain with the highest link count is determined to be the targeted

```

<HoneytokenRepository>/submit.jsp?site=<SiteName>
&field1=<FieldType>|<ValueHash1>|<ValueHash2>|...|<ValueHashN>
&field2=<FieldType>|<ValueHash1>|<ValueHash2>|...|<ValueHashN>
...
&fieldM=<FieldType>|<ValueHash1>|<ValueHash2>|...|<ValueHashN>

```

Figure 4.1: Honeytoken Repository Submission Format

site. If no valid external links exist of the page, the site is identified as “unknown”.

4.1.5 Uploading to the Honeytoken Repository

After the legitimate site has been identified and all requests have been sent to the phishing site, the honeytoken values and identified site domain are sent to the honeytoken repository. Figure 4.1 shows the request format for submitting to the repository. For each field, the classification (e.g. username) and a hash of each generated value are sent. Sending and storing hashed field values allows for future comparisons without storing the generated values themselves; should the repository ever become compromised, phishers will not be provided with any plaintext honeytoken values.

Currently, submissions to the honeytoken repository are sent out unencrypted; however, we believe it a trivial process to purchase a certificate and perform all repository requests over secure connections.

4.2 Server Web Application

The honeypot repository serves as a central location for generated honeypot tokens and provides a lookup service for legitimate sites. This service allows legitimate sites to check invalid credentials against the repository; presence in the repository indicates the credentials were created by Phish Feeder, and may be used to identify phishers using stolen credentials. Legitimate sites can use this information to temporarily block IP addresses or deter the phishers by other means.

The repository provides two simple services — a submission service which receives honeypot tokens from Phish Feeder clients and a lookup service for legitimate sites to perform credential lookups.

4.2.1 Receiving Honeypot Tokens

As each honeypot token submission is received, it is stored in the repository database. Each submission is given a unique id and associated with the target site, if one was provided. Each field classification and hash value are stored with the submission's id.

In order to deter malicious clients from spamming the repository, the IP address of each submission is temporarily stored. The honeypot repository limits each IP address to one submission every five minutes.

4.2.2 Honeypot Lookup Service

The honeypot repository provides a service which allows legitimate sites to lookup invalid credentials. Each lookup query includes a list of fields and value

```
<HoneytokenRepository>/lookup.jsp?site=<SiteName>  
  
    &field1=<FieldType>|<ValueHash>  
  
    &field2=<FieldType>|<ValueHash>  
  
    ...  
  
    &fieldM=<FieldType>|<ValueHash>
```

Figure 4.2: Honeytoken Repository Lookup Request Format

hashes. The lookup request format shown in Figure 4.2 is very similar to the honeytoken submission format; however, with the lookup request, only one value per field is included. The fields may be included in the request in any order — they do not need to be included in the same order as the original submission to the repository.

When the repository receives the request, it parses the information for each field and compares the values provided to those in the repository. All matching field classification and hash value pairs are returned and the strongest match is reported back to the querying server. The strength of the match is based on whether or not the target site, field classifications, and field values are matched. If multiple honeytoken submissions have matching field hash values, the submission with the higher number of matches is used; if multiple submissions have an equal number of matching hashes, submissions with matching target sites take precedence. In case of a tie, the most recent submission is used.

The response from the repository includes an overall confidence in the match, if the target site matched, and the match results for each of the fields; this report format presents the querying site with enough information to make an educated decision and is designed limit to the effects of collisions.

```

<Match-Strength>

<Match-Target-Site>

<Match-Field1>

<Match-Field2>

...

<Match-FieldN>

```

Figure 4.3: Honeytoken Repository Lookup Response Format

Match-Strength:	
– MATCH_FULL	All field information matched; confirmed Phish Feeder credentials
– MATCH_PARTIAL	Some field information matched; possible Phish Feeder credentials
– MATCH_NONE	No field information matched; not Phish Feeder credentials
Match-Target-Site:	
– TARGET_SITE_MATCH	Target site for data matched request site
– TARGET_SITE_UNKNOWN	Target site for data is unknown
Match-FieldN:	
– FIELD_MATCH	Both classification & hash value matched
– FIELD_CLASSIFICATION	Only field classification matched
– FIELD_VALUE	Only field hash value matched
– FIELD_NONE	No matches for this field

Table 4.3: Repository Lookup Response Values

Figure 4.3 displays the format for the lookup response, and the possible values for each of the fields in the response are presented in Table 4.3. The responses for the *Match-Strength* and *Match-Target-Site* fields are independent; both should be taken into consideration by the legitimate site. If the *Match-Strength* type is `MATCH_NONE`, then all field types will be `FIELD_NONE`; likewise, if the *Match-Strength* type is `MATCH_FULL`, all field types will be `FIELD_MATCH`. If a query response includes both `MATCH_FULL` and `TARGET_SITE_MATCH`, the querying site can be fairly certain that the credentials originated from Phish Feeder and take appropriate action.

It is possible that all of the fields will match (`MATCH_FULL`), but the target site will not (`TARGET_SITE_UNKNOWN`). This indicates that the credentials were generated for an unidentifiable target site. Phish Feeder only returns results for credentials with a matching or unknown target site; to prevent unintentional information leaks, the lookup service will not return matches for a different target site. Our evaluation in Section 5.2.2 demonstrates that Phish Feeder never identified a target site incorrectly — the target site was either properly identified or labeled as “unknown”.

The lookup service is intended for use by legitimate entities only; phishers with access to the lookup service could systematically query each set of received credentials to identify those generated by Phish Feeder. To prevent malicious use, a whitelist of trusted, legitimate IP addresses is maintained and only queries originating from these IP addresses are processed.

Chapter 5

Evaluation

We evaluated Phish Feeder on three fronts: the effectiveness of the generation algorithm, the correctness of the extension’s heuristics and behavior, and the extension’s performance. To test our improvements over previous credential generation algorithms, we conducted a user survey in which participants were asked to identify real credentials from a sets generated by either BogusBiter or Phish Feeder. We ran a variety of experiments to evaluate the correctness of each of our heuristics and to determine if the real requests within each set were deterministically identifiable. We also measured the performance overhead of the extension on both legitimate and phishing sites.

5.1 Generation Algorithm Effectiveness

We claim that the honeypot generation algorithm used by Phish Feeder is an improvement over its predecessors, specifically over the BogusBiter substitution algorithm. The goal of the algorithm is to effectively create fake credentials such that even a human has a hard time identifying the real credentials within a set.

Figure 5.1: User Survey Demographics

In order to evaluate this claim, we performed a user survey. Credential sets were created by running either the Phish Feeder or BogusBiter algorithm on a list of username/password pairs. Survey participants were presented with one of these lists and asked to choose the original credentials from each set.

The list of usernames was generated by crawling a popular online news site and a password list for the “John the Ripper” [3] security tool was obtained. Usernames and passwords were chosen at random from the two sets to create 25 username/password pairs. Credential Lists B and P were generated by running the BogusBiter and Phish Feeder generation algorithms, respectively, against the list of usernames/passwords pairs. Each credential list contained 25 sets of 10 credentials each — the nine generated credentials and the original credentials.

There were 24 total participants in the user study. All participants were members of the Computer Science or Computer Engineering departments of California Polytechnic State University, San Luis Obispo; the participants included one professor and 23 students of varying educational levels. Figures 5.1a and 5.1b provide histograms of the ages and education levels of the participants, respectively.

Each participant was randomly given either Credential List B or P and was

List B - BogusBiter	List P - Phish Feeder
Rebekkadavis97/Irchie	Rebekkadavis15/Bowery
Rebekkadavis87/Hrchie	Rebekkadavis14/Clevie
Rebekkadavis77/Grchie	Rebekkadavis13/Derwin
Rebekkadavis67/Frchie	Rebekkadavis12/Dirndl
Rebekkadavis57/Erchie	Rebekkadavis11/Dubber
Rebekkadavis47/Drchie	Rebekkadavis10/Milieu
Rebekkadavis37/Crchie	Rebekkadavis19/Ritual
Rebekkadavis27/Brchie	Rebekkadavis18/Signet
<i>Rebekkadavis17/Archie</i>	<i>Rebekkadavis17/Archie</i>
Rebekkadavis07/Zrchie	Rebekkadavis16/Stefan

Table 5.1: Credential Sets for User Survey Question #7

asked to identify the original credentials from the sets. For convenience, we will refer to those presented with Credential Lists B and P as Group B and Group P, respectively. Table 5.1 displays an example credential set for each algorithm.

We evaluated the survey responses based on the number of incorrectly chosen credentials; a survey participant choosing the incorrect credentials corresponds phishers using generated credentials instead of the originals. The results from the two groups were compared to discover which algorithm worked the best overall and which worked better in specific cases. Figure 5.2 shows the percent of incorrect responses for each generation algorithm. Overall, the Phish Feeder algorithm was more successful at hiding the real credentials; members of Group P chose the correct credentials from the set only 59.7% of the time, compared to 88.3% of the time for Group B.

The results show that the BogusBiter proved completely ineffective for 17 of the 25 credential sets; for these sets, members of Group B always correctly chose the real credentials. In contrast, the same was only true for 2 of the 25 credential sets created by the Phish Feeder algorithm. The real credentials from sets 8 and 9 — *Texasbig/firebird* and *Purple patriot/europe* — were always

Figure 5.2: User Survey Results — Percent Fake Credentials Chosen By Algorithm

identified correctly in both credential lists. These credentials contained multiple related words and compound words, both suboptimal choices for single character or word replacement.

The BogusBiter algorithm out-performed the Phish Feeder algorithm in only two cases, credential sets 6 and 14. In the case of credential set 6 — *Nug Master/Tanker* — Phish Feeder replaced *Master* and *Tanker* with less-common English words, while BogusBiter’s character replacement performed well and actually created other common words (*Tug*, *Rug*, *Banker*). With credential set 14 — *blh1616/gilles* — both the Phish Feeder and BogusBiter algorithms performed digit substitution on the username; however Phish Feeder used word substitution on the password (choosing *gill* as the replacement text), while BogusBiter performed character substitution. Both algorithms performed well, but the BogusBiter credential set received two more correct responses.

The Phish Feeder algorithm outperformed the BogusBiter algorithm in 23

out of 25 cases, and survey participants chose fake credentials almost four times as frequently when presented with the Phish Feeder credential list. The Phish Feeder algorithm significantly improves the chance that a phisher will attempt to use fake credentials before the real ones. As discussed in Section 3.2, this may allow legitimate sites to identify phishers before they have an opportunity to use the real credentials.

5.2 Extension Evaluation

We performed a variety of experiments to evaluate both the correctness and performance of the Firefox extension. The correctness tests focus on demonstrating that the user’s experience is not hampered, that the various identification techniques are adequate, and that the generated requests are indistinguishable from the real user request. The performance tests seek to demonstrate the overhead imposed by the Phish Feeder extension is reasonable for the end-user. All tests were run on an Intel i7 2.66GHz machine with 6GB of RAM and a 500GB HDD.

In several experiments, we refer to our Pond — a set of locally hosted phishing sites. We gathered 20 different phishing sites from the Phishtank Phish Archive [27]. The top 20 most recently reported sites were examined and downloaded locally. Phishers often operate several identical sites hosted at different locations. In order to evaluate our various heuristics against a broader input set, we avoided downloading and including multiple sites with the exact same layout and features. These sites were stored and hosted locally, as the lifetime of the average phishing site is too short to allow for repeatable testing. A report by the APWG claims that the average uptime of a phishing site is only 3 days [19].

google.com	yahoo.com	youtube.com	facebook.com	live.com
msn.com	wikipedia.org	blogger.com	myspace.com	wordpress.com
ebay.com	craigslist.com	flickr.com	amazon.com	aol.com
twitter.com	hi5.com	photobucket.com	imdb.com	skyrock.com

Table 5.2: Top 20 English-Speaking Sites

5.2.1 Correctness: False Positives

Phish Feeder does not intercept form submissions or generate honeytokens until the user navigates to a suspected phishing site. Infrequently, the user may navigate to a legitimate website which is incorrectly labeled as a phishing site. In this situation, the behavior of the Phish Feeder tool should not prohibit users from successfully navigating the site and should not appear to the server as a malicious attack. To test this, we compiled a list of the top 20 English-speaking sites from the Alexa Top 500 [5], as of June 2009. The list of sites accessed and tested with is shown in Table 5.2.

An account was created at each of the test sites. The Phish Feeder extension was configured to treat every page as a potential phishing site to emulate false positives from the Firefox phishing filter. We attempted to login to each of the test sites and recorded the observed behavior; at each of the 20 sites, we were able to successfully login without any adverse effects. This suggests that occasional phishing filter false positives will not cause Phish Feeder to adversely affect the user experience.

5.2.2 Correctness: Site Identification

Phish Feeder uses a basic site identification heuristic based on links to external domains to determine which legitimate site a phishing site is targeting. The target

site is determined to be the most frequently-linked domain on the page. To test this, we ran the site identification heuristic against the 20 sites in the Pond and manually verified the results.

Of the 20 phishing sites, the Phish Feeder heuristic was able to correctly determine the targeted site 40% (8) of the time. The rest of the target sites were unidentifiable either because the site contained no links (20%) or only relative links (40%). As discussed in Section 4.1.4, only absolute links are considered for the site identification heuristic. The heuristic obtained a 40% true positive rate and a 0% false positive rate — the heuristic never incorrectly identified the target site.

Associating generated credentials with their target sites increases the value of the honeypot repository and reduces lookup collisions. However, legitimate sites using the lookup service may still receive results for credentials which have no associated target site. Site identification — while beneficial — is not crucial to the success of the system. For this reason, a 40% identification rate is currently acceptable, although future work may be done to improve this percentage through addition of new or existing heuristics.

5.2.3 Correctness: Field Identification

Phish Feeder uses its field identification and classification heuristics to perform intelligent credential generation. We populated form data on each site in the Pond, submitted the form, and recorded the classification for each field. Most of the forms contained a simple username or email and password prompt, with a few exceptions. Figure 5.3 shows the breakdown of form inputs for the test sites.

Each field classification was recorded and manually verified. Phish Feeder

Figure 5.3: Form Input Categorization for Pond Test Sites

correctly identified the 76 total input fields on the 20 different forms, with zero false positives. A field identification was deemed correct if the field contained a type of sensitive data handled by Phish Feeder, and the algorithm correctly identified the field type. There were additional types of sensitive data contained within these forms which are not specifically addressed by Phish Feeder (e.g. date of birth or credit card expiration date); this may also serve as a point for future work.

5.2.4 Correctness: Position and Timing Tests

The success of Phish Feeder depends on its ability to properly hide the real request amongst the generated ones; if the phisher is able to automatically determine the real request from a set, the system will fail. We performed two separate tests to demonstrate that the real request is adequately hidden — a test which inspected the position of the real request within the set and a timing test which examined patterns of the time intervals between each request in the set.

Figure 5.4: Frequency of Real Request Position Within Request Set

We chose one phishing page from the Pond which contained a simple username/password form, and submitted the form 150 times; each form submission triggered Phish Feeder’s credential generation algorithm, which resulted in a total of 1500 requests to our server. We examined each set of 10 requests and determined the position of the real request within the set. Figure 5.4 displays the frequency of real request positions within their respective sets. While not perfectly uniform at this sample size, the graph shows that the real request position occurs with an even enough distribution that no request can be assumed fake based on its position within the set.

To validate that the request creation did not introduce any timing oddities, we recorded the time each request resulting from the last 50 form submissions was received. The time intervals between each request within a set were analyzed and compared to the intervals for the requests directly before and after the real request. The results of this comparison are available in Table 5.3. The real requests show no obvious timing patterns which could distinguish them from the fake requests.

	Overall	Real
Average	7.85 ms	7.77 ms
Median	8 ms	8 ms
Std. Dev.	0.73 ms	0.70 ms

Table 5.3: Time Intervals Between Requests – All vs. Real Requests

We have demonstrated that the real request can not be distinguished by either its position within the request set or the time delay between requests. As all properties of the request itself are duplicated between the requests — with the exception of the generated credentials replacing the originals — we observe that the only way for a phisher to distinguish the real request is to manually identify which credentials are real and which are fake.

5.2.5 Performance Testing

In the preceding evaluations, we have shown that the Phish Feeder extension functions as intended; however, if the extension disrupts the user experience through sluggish performance, its usefulness becomes quite limited. What follows are a series of experiments to measure the overhead incurred by installing and using the Phish Feeder extension. We observed the overhead in generating honeytokens and the overall delay in form submissions.

We logged the time necessary to gather form information and generate honeytokens from two of the sites from the Pond. The first site contained a basic username/password login form; the second contained fifteen fields of different classifications. We found gathering form information and generating honeytokens to take between 1–5 milliseconds, depending on the site structure and complexity.

To gauge total overhead, we measured request times at two locally hosted sites

Figure 5.5: Average Request Times to Test Sites With And Without Extension Enabled

— one phishing and one legitimate — both with and without the Phish Feeder extension enabled. For the legitimate site, this measures the overhead incurred from Phish Feeder’s “quiet state”, in which it is monitoring requests and waiting for the user to navigate to a suspected phishing site. For the phishing site, this measures the overhead of generating honeytokens and creating requests. We chose a PayPal site from the Pond as our sample phishing site, and a local copy of the actual PayPal login page as our legitimate site. Each site’s login form was submitted 5 times and the average request time was recorded. The Extended Statusbar Firefox extension was used to record request times [23].

Figure 5.5 displays the results from the performance tests. On the phishing site, as requests are being created and sent, Phish Feeder introduced marginal overhead. In the more common case of browsing legitimate sites, the observed overhead is negligible. We feel comfortable that for the large majority of users’ web browsing, the overhead will be unnoticeable; in the less common case of submitting to a phishing site, the overhead is reasonable given the work performed.

Chapter 6

Conclusions

Phishing continues to prove successful for the phishers; hundreds of new phishing sites are launched daily in an attempt to gather large sets of user credentials. Anti-phishing groups, corporations, and researchers continue to develop anti-phishing technologies to detect phishing sites and emails, alert users of phishing attacks, and to generally reduce the effectiveness of phishers. This collection of anti-phishing tools and technologies continues to increase in size and complexity as phishing sites become increasingly realistic and convincing.

In this paper, we have presented Phish Feeder, an anti-phishing tool which uses carefully-crafted fake credentials to clutter phishers' databases, hide real users' credentials, and serve as a potential phisher identification tool. Once the user submits a form on a suspected phishing site, the credential generation algorithm creates a set of fake credentials based on the user's real credentials, then submits the entire credential set to the phishing site. The generated credentials are then sent to a honeypot repository to facilitate future querying from legitimate sites.

Users who install the Phish Feeder extension are protected on any suspected phishing site and their user experience remains unaffected on all other sites. Our performance testing showed that the monitoring Phish Feeder performs on legitimate sites introduces a negligible overhead. Our testing also indicates that Phish Feeder will not adversely effect the user experience when phishing detection encounters a false positive.

Our credential generation algorithm intelligently generates credentials based on users' real credentials. We conducted a survey in which users were presented with sets of credentials and asked to identify the real credentials within each set; users were able to chose the real credentials less than 60% of the time, a significant improvement over previous credential generation algorithms.

The generated credentials are sent to the phishing site alongside the real request. Our testing shows no identifiable heuristic for determining the real request from the set based on position or timing.

We suggest a central honeypot repository to store the generated credentials and provide a lookup service for legitimate sites. This service allows legitimate sites to determine if invalid credentials were generated by Phish Feeder. This information can potentially be used to identify phishers and protect end-users.

Our evaluation confirms that Phish Feeder represents an improvement over previous work and is a significant contribution in the area of credential-based anti-phishing tools.

6.1 Future Work

Phish Feeder represents a step forward in the area of fake-credential-based anti-phishing tools. While it has met its goals, there is room for improvements and additions to the various heuristics present in the tool.

Phish Feeder identifies target sites based on a simple link-based heuristic. In our evaluation, this heuristic was able to identify 40% of the target sites. Target site identification allows the credentials to be associated with the correct site in the repository, which can increase lookup response confidence and reduce collisions. Various additions may be made to this heuristic to increase the success rate, including image and site layout recognition [10] [16].

Phish Feeder currently identifies and handles usernames, passwords, social security numbers, credit card numbers, and email addresses. Phish Feeder also generates honeypot values for any field containing digits. Future work could improve heuristics for identifying existing field classifications, or add new classifications to represent sensitive information not included in the above list.

While Phish Feeder’s credential generation algorithm represented a significant improvement over the previous work, more than half the credentials were still properly identified by users. Phish Feeder’s word-based generation algorithm performed inadequately when the credentials contained multiple correlated words. This may be improved by updating the generation algorithm to choose words related to the chosen replacement word, or to give higher preference to more-frequently used English words. Expanding the algorithm to work in languages other than English would also increase the effectiveness of the honeypot creation.

Bibliography

- [1] Anti-phishing working group, April 2009. <http://www.antiphishing.org/>.
- [2] Gnu aspell, June 2009. <http://www.aspell.net>.
- [3] John the ripper password cracker, June 2009 .
<http://www.openwall.com/john>.
- [4] B. Adida. Beamauth: two-factor web authentication with a bookmark. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 48–57, New York, NY, USA, 2007. ACM.
- [5] Alexa top sites, Jun 2009. <http://www.alexa.com/topsites>.
- [6] Y. Cao, W. Han, and Y. Le. Anti-phishing based on automated individual white-list. In *DIM '08: Proceedings of the 4th ACM workshop on Digital identity management*, pages 51–60, New York, NY, USA, 2008. ACM.
- [7] M. D. Center. Xpcom. March 2009 .
<https://developer.mozilla.org/en/XPCOM>.
- [8] M. Chandrasekaran, R. Chinchani, and S. Upadhyaya. Phoney: mimicking user response to detect phishing attacks. *World of Wireless, Mobile and Multimedia Networks, 2006. WoWMoM 2006. International Symposium on a*, pages 5 pp.–672, 0-0 2006.

- [9] J. Chen and C. Guo. Online detection and prevention of phishing attacks. *Communications and Networking in China, 2006. ChinaCom '06. First International Conference on*, pages 1–7, Oct. 2006.
- [10] N. Chou, R. Ledesma, Y. Teraguchi, D. Boneh, and J. Mitchell. Client-side defense against web-based identity theft. *11th Network and Distributed System Security Symposium (NDSS)*, 2004.
- [11] L. Cranor, S. Egelman, J. Hong, and Y. Zhang. Phinding phish: An evaluation of anti-phishing toolbars. *14th Network and Distributed System Security Symposium (NDSS)*, 2007.
- [12] R. Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In *SOUPS '05: Proceedings of the 2005 symposium on Usable privacy and security*, pages 77–88, New York, NY, USA, 2005. ACM.
- [13] DSLReports.com. Phish tracker, April 2009 .
<http://www.dslreports.com/phishtrack>.
- [14] S. Egelman, L. F. Cranor, and J. Hong. You’ve been warned: an empirical study of the effectiveness of web browser phishing warnings. In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 1065–1074, New York, NY, USA, 2008. ACM.
- [15] A. Emigh. Online identity theft: Phishing technology, choke-points and countermeasures. *ITTC Report on Online Identity Theft Technology and Countermeasures*, October 2005.
www.anti-phishing.org/Phishing-dhs-report.pdf.
- [16] A. Y. Fu. Detecting phishing web pages with visual similarity assessment based on earth mover’s distance (emd). *IEEE Trans. Dependable Se-*

- cur. Comput.*, 3(4):301–311, 2006. Senior Member-Wenyin, Liu and Senior Member-Deng, Xiaotie.
- [17] S. Garera, N. Provos, M. Chew, and A. D. Rubin. A framework for detection and measurement of phishing attacks. In *WORM '07: Proceedings of the 2007 ACM workshop on Recurring malware*, pages 1–8, New York, NY, USA, 2007. ACM.
 - [18] M. Gertner. Creating custom firefox extensions with the mozilla build system, Jan 2006. https://developer.mozilla.org/En/Creating_Custom_Firefox_Extensions_with_the_Mozilla_Build_System.
 - [19] A.-P. W. Group. Phishing activity trends: Report for the month of january, 2008. January 2008. http://www.apwg.org/reports/apwg_report_jan_2008.pdf.
 - [20] A.-P. W. Group. Global phishing survey: Trends and domain name use 2h2008. May 2009. http://www.apwg.org/reports/APWG_GlobalPhishingSurvey2H2008.pdf.
 - [21] C. Jackson, D. R. Simon, D. S. Tan, and A. Barth. An evaluation of extended validation and picture-in-picture phishing attacks. In *In Proceedings of Usable Security (USEC07)*, 2007.
 - [22] E. Kirda and C. Kruegel. Protecting users against phishing attacks with antiphish. *Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International*, 1:517–524 Vol. 2, July 2005.
 - [23] S. Kustudic. Extended statusbar, September 2008. <https://addons.mozilla.org/en-US/firefox/addon/1433>.

- [24] P. Likarish, E. Jung, D. Dunbar, T. Hansen, and J. Hourcade. B-apt: Bayesian anti-phishing toolbar. *Communications, 2008. ICC '08. IEEE International Conference on*, pages 1745–1749, May 2008.
- [25] Mozilla. Firefox phishing and malware protection, April 2009. <http://www.mozilla.com/en-US/firefox/phishing-protection/>.
- [26] B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. *Financial Cryptography*, 4107 of Lecture Notes in Computer Science:1–19, 2006.
- [27] Phishtank. Phishtank — join the fight against phishing, June 2009. <http://www.phishtank.com/>.
- [28] C. E. Reid. History of phishing. February 2009. <http://www.allspammedup.com/2009/02/history-of-phishing/>.
- [29] T. Ronda, S. Saroiu, and A. Wolman. Itrustpage: a user-assisted anti-phishing tool. In *Eurosys '08: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*, pages 261–272, New York, NY, USA, 2008. ACM.
- [30] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.
- [31] S. Sheng, B. Wardman, G. Warner, L. Cranor, J. Hong, and C. Zhang. An empirical analysis of phishing blacklists. *Conference on Email and Antispam*, July 2009.
- [32] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *CHI '06: Proceedings of the SIGCHI conference*

- on *Human Factors in computing systems*, pages 601–610, New York, NY, USA, 2006. ACM.
- [33] M. Wu, R. C. Miller, and G. Little. Web wallet: preventing phishing attacks by revealing user intentions. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 102–113, New York, NY, USA, 2006. ACM.
- [34] T. Wu. The secure remote password protocol. In *In Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1997.
- [35] K.-P. Yee and K. Sitaker. Passpet: convenient password management and phishing protection. In *SOUPS '06: Proceedings of the second symposium on Usable privacy and security*, pages 32–43, New York, NY, USA, 2006. ACM.
- [36] C. Yue and H. Wang. Anti-phishing in offense and defense. *Computer Security Applications Conference, 2008. ACSAC 2008. Annual*, pages 345–354, Dec. 2008.
- [37] Y. Zhang, J. I. Hong, and L. F. Cranor. Cantina: a content-based approach to detecting phishing web sites. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 639–648, New York, NY, USA, 2007. ACM.