Detection and Prevention of Sophisticated Cyberattacks

by

Penghui Zhang

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved March 2022 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Co-Chair
Adam Doupé, Co-Chair
Adam Oest
Alexandros Kapravelos

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

Despite extensive research by the security community, cyberattacks such as phishing and Internet of Things (IoT) attacks remain profitable to criminals and continue to cause substantial damage not only to the victim users that they target, but also the organizations they impersonate. In recent years, phishing websites have taken the place of malware websites as the most prevalent web-based threat. Even though technical countermeasures effectively mitigate web-based malware, phishing websites continue to grow in sophistication and successfully slip past modern defenses. Phishing attack and its countermeasure have entered into a new era, where one side has upgraded their weapon, attempting to conquer the other. In addition, the amount and usage of IoT devices increases rapidly because of the development and deployment of 5G network. Although researchers have proposed secure execution environment, attacks targeting those devices can often succeed. Therefore, the security community desperately needs detection and prevention methodologies to fight against phishing and IoT attacks.

In this dissertation, I design a framework, named CrawlPhish, to understand the prevalence and nature of such sophistications, including cloaking, in phishing attacks, which evade detections from the anti-phishing ecosystem by distinguishing the traffic between a crawler and a real Internet user and hence maximize the return-on-investment from phishing attacks. CrawlPhish also detects and categorizes client-side cloaking techniques in phishing with scalability and automation. Furthermore, I focus on the analysis redirection abuse in advanced phishing websites and hence propose mitigations to classify malicious redirection use via machine learning algorithms. Based on the observations from previous work, from the perspective of prevention, I design a novel anti-phishing system called Spartacus that can be deployed from the user end to completely neutralize phishing attacks. Lastly, inspired by Spartacus, I propose iCORE, which proactively monitors the operations in the trusted execution environment to identify any maliciousness.

ACKNOWLEDGEMENTS

My research would not have been possible without the tremendous support of my advisors, fellow students, and industry collaborators.

Dr. Ahn, thank you so much for picking me up when I was looking for my research area at the beginning of my Ph.D. career. Thanks to you, I was able to join in SEFCOM and work with talented students and professors, which was a new chapter of my life.

Dr. Doupé, I appreciate your support when I decided to explore other possibilities in different topics in the cybersecurity area. I still remember your encouragement: "Go ahead, you definitely can explore different research topics when you are a Ph.D. student". Because of you, I was able to work on mitigations against sophisticated phishing websites and succeed.

Dr. Oest, I am so fortunate that we could know each other at the first day in SEFCOM. We both were TA for the Computer Forensics class. Thank you so much for bringing me insights about web security and state-of-the-art research topics you have been working on. I also appreciate your effort, advice, and suggestions about the projects we've been together researching on. I am so happy that I can have a friend and an advisor like you.

Dr. Kapravelos, thank you for introducing me to the world of fingerprinting. Without your previous work and your advice, I would have not been able to mitigate cloaking techniques in phishing attacks.

I am so honored to be one member of SEFCOM, and I would like to thank all my co-authors and fellow students who have worked with me on projects over the past years. I would also like to thank the professors for supporting the exciting trips and activities that we have had outside of the lab, and for their dedication to driving the lab's spectacular growth.

Last but not the least, I hope to thank my wonderful wife, Xin, for supporting me during countless days when I struggled with work and bringing me suggestions and knowledge

from your perspective, out of the case. Additionally, I wish to say a “thank you” for my wonderful parents, who guide me to achieve the highest degree human can ever get.

TABLE OF CONTENTS

Page

LIST OF TABLES ix

LIST OF FIGURES xi

CHAPTER

1 INTRODUCTION 1

2 BACKGROUND 11

2.1 Cloaking Techniques in Phishing 11

2.2 Challenges in Analyzing Client-side Cloaking 13

2.3 Redirection 14

2.4 IoT Platform and Security 15

2.4.1 Monitoring Mechanisms 15

2.4.2 ARM TrustZone Architecture 17

3 UNDERSTANDING AND CATEGORIZING SOPHISTICATIONS IN PHISHING 21

3.1 Crawling & Pre-processing 23

3.2 Feature Extraction 24

3.3 Cloaking Detection 26

3.4 Cloaking Categorization 28

3.5 Evaluation: Detection of Cloaked Phishing Websites 30

3.6 Evaluation: Cloaking Categorization 32

3.6.1 Categorization of Cloaking Types 33

3.6.2 Accuracy of Cloaking Categorization 37

3.6.3 Grouping of Implementations 38

3.6.4 Trends in Cloaking Usage 41

3.7 Evaluation: Impact of Cloaking techniques 43

CHAPTER Page

3.7.1 Effectiveness Against Anti-Phishing Entities 43
3.7.2 Hampering Victim User Traffic 45
3.7.3 Responsible Disclosure 46
3.8 Countering Client-side Cloaking Techniques 47
3.9 Related work 48
3.10 Conclusion 50
4 UNDERSTANDING THE SCALY UNDERBELLY OF REDIRECTION CHAINS IN PHISHING 51
4.1 PHISHPATH Overview 54
4.2 Data Collection 55
4.3 Redirections in Phishing Campaigns 57
4.3.1 The Prevalence of Redirection 57
4.3.2 Anti-phishing System Performance 58
4.3.3 Benign and Malicious Redirection Chains 59
4.3.4 Redirection with Cloaking Techniques 61
4.4 Phishing Redirection Behavior 61
4.4.1 Redirection Domain Reuse 62
4.4.2 Malicious URL Shortening 64
4.4.3 Redirection Chain Length 65
4.4.4 Open Redirect Vulnerability Exploitation 66
4.4.5 Miscellaneous 67
4.5 Impact of Malicious Redirection 68
4.6 Machine Learning Based Classification 70
4.6.1 Feature Selection 71

CHAPTER Page

4.6.2 Model Selection 72
4.6.3 Early Classification of Sub-chains 74
4.6.4 Complementing Current Systems 76
4.7 Blacklisting Less, Blocking More 77
4.8 Discussion 78
4.8.1 How Did We Get Here? 78
4.8.2 Where Do We Go? 80
4.8.3 Limitations 81
4.8.4 Responsible Disclosure 83
4.9 Related Work 83
4.10 Conclusion 84
5 SPARTACUS: CLOAK USERS AGAINST CLOAKED PHISHING WEBSITES 86
5.1 Prevalence of Fingerprinting-based Cloaking 86
5.2 Design 87
5.2.1 Overview 88
5.2.2 Visit Pre-filters 89
5.2.3 Bot Profile Mutator 90
5.2.4 Suspicious Content Classification 91
5.3 Privacy 92
5.3.1 Privacy Information 93
5.3.2 Privacy Consent and Protection 93
5.4 Evaluation 94
5.4.1 Effectiveness 95

CHAPTER Page

5.4.2 Effectiveness of Trigger Words 96
5.4.3 Effectiveness of Proxy Server Option 97
5.4.4 Efficiency and Latency 98
5.4.5 Impact on Benign Web Sites 100
5.5 Ecosystem Support 107
5.6 Mitigating Server-side Cloaking 109
5.7 Countermeasures to Spartacus 111
5.7.1 Using Other Cloaking Techniques 111
5.7.2 Emerging Phishing Based on Spartacus 112
5.8 Limitations 114
5.8.1 Spartacus Design 114
5.8.2 Spartacus Deployment and Evaluation 115
5.9 Related Work 115
5.10 Conclusion 117
6 ICORE: CONTINUOUS AND PROACTIVE EXTROSPECTION ON MULTI-CORE IOT DEVICES 120
6.1 Assumptions and Threat Model 120
6.1.1 Assumptions 120
6.1.2 Threat Model 120
6.2 System Design 121
6.2.1 Initialization of iCORE 122
6.2.2 Continuous and Proactive Extrospection 124
6.3 Implementation 126
6.3.1 Core Initialization 127

CHAPTER Page

6.3.2 Memory Acquisition 128
6.3.3 Continuous and Proactive Extrospection 130
6.4 Evaluation 131
6.4.1 Effectiveness of iCORE 131
6.4.2 Performance with iCORE 132
6.5 Discussion and Future work 134
6.5.1 Response After Detection 134
6.5.2 Potential Threats Against iCORE 135
6.5.3 Extension of iCORE 136
6.6 Related Work 137
6.7 Conclusion 139
7 CONCLUSION 140
7.1 Attacks Can Be mitigated 140
7.2 Remaining Challenges 141
REFERENCES 143

LIST OF TABLES

Table		Page
3.1	Summary of the Client-side Cloaking Technique Types Identified in This Work.	22
3.2	Accuracy of Cloaking Detection by CrawlPhish.	31
3.3	Overview of the Number of Distinct Groups of Cloaking Code Implementations in the APWG and Public Datasets.	35
3.4	Cloaking Technique Types in the APWG Dataset, as Detected by Crawlphish.	41
3.5	Cloaking Technique Types in the Public Dataset (September to December 2019), as Detected by CrawlPhish.	42
3.6	Top Brands Targeted by Cloaked Phishing Websites in the APWG Dataset.	43
3.7	Experimental Results on the Effect of Cloaking Techniques on Users' Ability to See Phishing Content.	45
4.1	Percentage of URLs in Malicious (APWG and Phishing Email) and Benign Data Sets Demonstrating Each Feature.	60
4.2	Top URL Shortening Providers Abused by Phishing Attacks in Our Datasets.	62
4.3	Abuse Reporting Types of Top URL Shortening Services.	67
4.4	Blacklist/Detection Speed by the Ecosystem in hh:mm Break-down by Redirection Techniques in the Current Commodity Dataset.	69
4.5	Evaluation Metrics of Evaluation of Our Classification on Websites with Redirection.	70
4.6	12 Features That PHISHPATH Uses for Prediction and the Output Label of the Prediction.	71
5.1	Top 10 Sensitive Words Appeared in the Phishing Kits Examined by Us.	87
5.2	Top 10 Trigger Words That Evaded Phishing Content.	97

Table Page

5.3 Coarse-grained Experiment Result Of Spartacus with and Without Applying Logic of Mutating HTTP Profile. 100

5.4 Fine-grained Experiment Result Of Spartacus Visit, Compared with the Result of Default Browser Visit. 103

5.5 Evaluation Metrics of Evaluation of Our Classification on Websites with Redirection. 110

5.6 Exthouse Metrics of Top 10 Chrome Extensions [1] along with Spartacus When Visiting Benign and Malicious Websites. 118

6.1 Performance Overhead When Checking Different Size of Static Memory Area. 132

6.2 Comparison of the Related Works with ICORE. 135

LIST OF FIGURES

Figure		Page
1.1	The Volume of Phishing Attacks Have Been Surging Year over Year [2].	10
2.1	Typical Operation of Server-side Fingerprinting-based Cloaking in Phishing Web Sites.	12
2.2	Typical Operation of Client-side Cloaking in Phishing Websites.	12
3.1	CrawlPhish Architecture.	22
3.2	ROC Curves to Select Thresholds for Cloaking Detection and Cloaking Type Categorization.	28
3.3	Initial and Force-executed Appearance of a Phishing Website With *Pop-up* Cloaking.	33
3.4	A Phishing Website with the Evolved Pop-up (Notification) Cloaking Technique, in Which the Web Page Directs Human Visitors to Click on the "allow" Button by Showing an Arrow.	34
3.5	Appearance of a Phishing Website with The *Cookie* Cloaking Technique.	36
3.6	CDF of Implementation Groups for All Phishing Websites in the Apwg Dataset.	39
4.1	Example of a Phishing Redirection Chain: The lure phase (Green), intermediate phase (Blue), and landing phase (Red).	52
4.2	PHISHPATH Analysis Flow.	54
4.3	Example of Redirection Chain with Re-used Hops.	57
4.4	Comparisons Between Malicious and Benign Redirection Implementation to Quantify Indicators.	59
4.5	Most Used Redirection Domains in the Commodity Dataset. Long Domain Names Are Shortened by First Removing Their Tlds and Then Leaving the First 18 Characters.	63

Figure Page

4.6 A Novel Feature in *linktr.ee* (`https://linktr.ee/PayPal`) That Requires User Interaction. 65
4.7 Feature Importance Indicated by Decrease on ROC-AUC. The Decrease Is Calculated by Subtracting Permutated ROC-AUC from Baseline ROC-AUC (0.954). Predictive Features Are Mainly from Advanced Feature Groups. 72
4.8 ROC Curves of PHISHPATH with and Without Sub-chain Mechanism. 74
5.1 Spartacus Architecture and Its Workflow. 88
5.2 Web Page Content from Default and Spartacus Browser Visits for a Cloaked Phishing Web Site. 95
5.3 Venn Diagram Describing the Ability of Spartacus and Current Anti-phishing Systems Against Phishing Web Sites. The Unit Is the Amount of Phishing Web Sites. 107
5.4 CDF of Blacklist/Detection Time Within *24 Hours* of Current Anti-phishing Systems Against Detected Phishing URLs Evaded and Not Evaded by Spartacus. 109
5.5 CDF of Blacklist/Detection Time of Current Anti-phishing Systems Against Detected Phishing URLs Evaded and Not Evaded by Spartacus. 110
5.6 Simplified PHP Code Snippet of Fingerprinting-based Cloaking in a Phishing Kit, Checking IP, Hostname, and User-Agent. 118
5.7 Difference Due to the Shape of Buttons. 119
5.8 Difference Due to Popup. 119
6.1 ICORE Architecture. 121
6.2 Core Initialization with ICORE. 122

Figure Page

6.3 Extrospection of ICORE. 125
6.4 Evaluation Result in SPEC CPU2017. 132

Chapter 1

INTRODUCTION

Cyber attacks such as phishing and IoT attacks have reached record levels of volume in recent years and continue to cause substantial damage—both direct and collateral—to Internet users and victim organizations. According to the analysis of Google Safe Browsing shown in Figure 1.1, phishing attacks have been increasing in recent years and have replaced malware websites to be the most popular online attack. Pandemics such as COVID-19 can also be exploited by phishers to deploy their attacks in large scale, which is observed also in Figure 1.1 as a spike in February 2020. Despite advancements in detection and mitigation efforts, the fight against phishing continues to be a cat-and-mouse game [3, 4]. Sophisticated phishing websites implement evasion techniques to delay or avoid detection by automated anti-phishing systems, which, in turn, maximizes the attackers' return-on-investment [5]. Such evasion—known as *cloaking*—typically seeks to determine if a visitor to the website is a bot, and shows benign content if so. The danger posed by successful evasion is exacerbated by these websites' efforts to steal more than just usernames and passwords: today's phishing attacks seek to harvest victims' full identities, which can cause wider damage throughout the ecosystem and is more challenging to effectively mitigate [6]. In addition, as the development of 5G network, the number of IoT devices increases rapidly in recent years. To protect security sensitive operations, platforms such as ARM have proposed a secure execution environment (or secure world). However, the secure world is a slave of the normal world and only does security operations that the normal world asks. Therefore, the secure world is passively protecting the IoT devices even with high privileges. Similar to the anti-phishing ecosystem, attackers who compromise the normal world can easily disable the secure world.

Thwarting criminals' evasion efforts is, thus, an important problem within the security community, as timely detection and continuous monitoring is the key to successful mitigation. Prior research has characterized server-side cloaking techniques used by phishing websites [4, 7, 8] and showed that they can defeat key ecosystem defenses such as browser-based detection [5]. However, the nature and prevalence of advanced cloaking techniques, such as those implemented on the *client-side* using JavaScript, is poorly understood. Client-side cloaking can be particularly dangerous because it enables the implementation of complex interactions with potential victims.

Moreover, today's cybercrime is fueled by an underground economy that is capable of providing a full suite of commoditized services, such as web hosting [9], e-mail spamming [10], and cash-out [11], which greatly lowers the barrier to entry for phishing attacks and other mischief. Phishers can retrieve and exploit defaced servers from blackmarkets for malicious purposes. For this reason, servers which are defaced have subsequently been linked to future attacks such as phishing and malware [12]. Prior work has shown a clear temporal correlation between defacements and phishing, but prediction been limited in scope to phishing sites with the same domain as the defacement [13] or *existing* web pages which subsequently turn malicious [14].

The ultimate desire of phishers is to maximize the profits from attacks by extending the lifespan of their phishing websites as long as possible and to lure real human visitors to submit credentials. To destroy phishing attack completely, I need an anti-phishing system that neutralize phishing content from the user end.

On the other hand, researchers have proposed two classes of monitoring mechanisms to monitor potential malicious activities for IoT devices [15]. First, "in-the-box" approaches, refer to the security tools that reside in the OS kernel. Second, "out-of-the-box" approaches, are the tools who stay outside of the kernel. However, both of the mechanisms have limitations. For example, "in-the-box" security tools can be easily compromised if

the OS kernel is under attacker's control. Also, "out-of-the-box" tools rely on vulnerable hypervisors and external hardware components. The shortcomings make it difficult to keep the security tools safe or to reduce the cost of virtualization and additional hardware deployment. Recently, hardware isolated execution environments, such as ARM TrustZone [16], AMD SVM [17], and Intel TXT [18], were proposed to provide a trusted environment for secure execution outside of the normal execution environment. Among the isolated architectures, ARM TrustZone is exploited most on IoT devices [19]. In the existing ARM TrustZone paradigm, the secure world is designed as a slave of the normal world, because it only executes the operations that the normal world requests. This architecture facilities the researchers to develop security tools that reside outside of the monitored system, such as in TZ-RKP [20] and SPROBES [21]. However, to implement such security tools, the developers have to make invasive changes in the normal world OS kernel to interrupt certain functionalities. Besides, these methodologies burden the normal world OS kernel by frequently performing a world switch. Additionally, due to the traditional relationship between two worlds, the functionalities of the security tools can be never invoked if the attacker who compromises the normal world chooses not to. Therefore, IoT devices still encounter with potential attacks even though such security tools reside.

Therefore, in this dissertation, I propose both pain relief pills and a vaccine against phishing-virus as follows. I design a framework named *CrawlPhish* to first understand the nature and prevalence of client-side cloaking techniques in phishing and then to detect and categorize cloaked phishing websites automatically. *CrawlPhish* is a robust framework that harvests the source code of live, previously reported phishing websites in the wild and automatically detects and categorizes the client-side cloaking techniques used by these websites. By efficiently adapting advanced program analysis techniques inspired by prior research of JavaScript malware [22, 23, 24, 25], my framework can not only identify the semantics of these cloaking techniques, but also track the evolution of code written by

specific phishing kit authors [26]. I use the CrawlPhish framework to perform a large-scale evaluation of the landscape of client-side cloaking used by phishing websites. In total, over a period of 14 months from mid-2018 to mid-2019, I collected and thoroughly analyzed 112,005 phishing websites. I measured the prevalence of client-side cloaking techniques within these websites and discovered that 35,067 (31.3%) use such cloaking. Thereof, I identified 1,128 groups of related implementations which I believe stem from distinct threat actors. Moreover, I observed that the percentage of phishing websites with client-side cloaking grew from 23.32% in 2018 to 33.70% in 2019. To understand why client-side cloaking is used so frequently, I characterize the ways in which it functions, and I define eight different types of evasion techniques in three high-level categories: *User Interaction*, *Fingerprinting*, and *Bot Behavior*. Respectively, the techniques within these categories require human visitors to perform a task, profile the visitor based on various attributes, or exploit technical differences between browsers used by crawlers and real browsers. I evaluated CrawlPhish and found that it could detect the presence of cloaking with low false-positive (1.45%) and false-negative (1.75%) rates, while requiring an average of 29.96 seconds to analyze each phishing website. Once CrawlPhish has detected cloaking, it can then reliably categorize the *semantics* of the cloaking technique by using both static and dynamic code features. Finally, to show that client-side cloaking poses a real-world threat, I deploy 150 carefully-controlled artificial phishing websites to empirically demonstrate that all three categories of evasion can successfully bypass browser-based detection by Google Chrome, Microsoft Edge, and other major web browsers. I also demonstrate that these websites remain accessible to potential human victims. As a result, I disclosed my findings to the aforementioned browser developers, who are working to improve the timeliness of the detection of the corresponding phishing websites. My analysis furthers the understanding of the nature of sophisticated phishing websites. In addition, the CrawlPhish framework can be deployed to continuously monitor trends within complex evasion techniques while

identifying new types of techniques as they are introduced by attackers. My methodology can not only directly help address gaps in the ecosystem's detection of sophisticated phishing websites, but can also aid in the development of attributes to improve existing anti-phishing mitigations such as browser-based detection.

I then propose PHISHPATH to perform the first large-scale analysis on redirection use in the modern phishing ecosystem by collecting and analyzing phishing emails and live websites. I ran my study for over a year from 2020 to 2021, collecting and analyzing 136,791 distinct phishing URLs from the Anti-Phishing Working Group (APWG), PhishTank, and victim-reported phishing emails from a frequent phishing-targeted organization. I discovered that *phishers are using redirection techniques* and that *current anti-phishing systems are not capable of effectively detecting phishing campaigns that use redirection links*. I found extensive redirection use in phishing, with 34% of phishing attacks making use of redirection techniques. I identified a common pattern in the design of redirection chains, which I diagram in Figure 4.1: a *lure phase*, when a URL is embedded in phishing emails or social media posts that trick victims into clicking, after which the victim's HTTP request is redirected through the *intermediate phase*, where phishers identify anti-phishing bots and divert them to legitimate websites. Finally, after passing the server-side and client-side cloaking techniques in the intermediate phase, the visitors will see the phishing content in the *landing phase*. While prior work studied phishing from the perspective of the lure phase and landing phase, I found that the intermediate phase is critical in the evasion of phishing detection (Section 4.4.1). I found that attackers employ URL shortening services to help evade detection systems, including shorteners that *require user interaction* with a web page, rendering detection systems ineffective. Such phishing links were active (and not blacklisted) five months after initial detection. Moreover, I found that phishers exploit open redirect services in LinkedIn and Google to hide their malicious URLs. Aside from the use of the intermediate phase for cloaking, I found that it also granted attackers ad-

ditional flexibility in adapting to blacklist-based defenses. Because anti-phishing systems only blacklist specific URLs instead of domains, I observed phishers generating new URLs under the same domain to replace previously-blacklisted hops and keep the chain accessible for *as long as seven months*. In fact, I found that over 30% of phishing redirection chains *reused* domains in their intermediate layers. I also measured the performance of anti-phishing blacklisting, online URL scanning, and feature-rich machine learning classification against redirection- and non-redirection-enhanced phishing attacks. Facing redirection phishing websites, these methodologies show degradation on the performance, compared with that against non-redirection ones. The difference in blacklisting [27] and online URL scanning [28] latency between the two groups is 6 hours and 42 minutes on average. Additionally, a state-of-the-art machine learning classifier [29] has a high false-negative rate (10.82%) and a low F1 score (0.87) when categorizing redirection phishing websites, compared with the 0.37% of false-negative rate and the 0.99 F1 score recorded in the previous work. These results demonstrate that redirection techniques effectively delay detection and classification. Based on my analysis results, I propose two mitigation techniques. First, blacklist-based anti-phishing systems can block commonly reused redirection domains instead of only specific URLs, with few resulting false positives. This defense would end the currently trivial bypass of blacklists by attackers and reduce phishing attack profitability due to increasing malicious domain "burn rate." Next, I design a machine learning model based on attributes of redirection chains used in phishing websites to classify such sites without relying on content-based features in often-stealthy landing pages. The PHISHPATH classifier can identify phishing pages with only redirection information with high accuracy (90.76%) and a high F1 score (0.92). Finally, I discover that many entities are involved in redirection use in phishing. Based on this finding, I propose several ecosystem mitigations, including the tracking of redirection data, using redirection data to classify advanced phishing as well as locate lures, and restricting redirection services without identity

information.

Furthermore, I consider the anti-phishing problem from a different angle and strike at the core reason behind the effectiveness of server-side cloaking techniques: rather than attempting to detect server-side cloaking through improved analysis tools, I instead *make the legitimate users themselves look like anti-phishing crawlers, so that server-side cloaked phishing pages will decline to phish them*. In this way, users can evade phishing content in real time, without prior knowledge of the phishing web site, by *leveraging*, instead of fighting, the cloaking functionality in the phishing site itself. To realize this idea, I propose Spartacus, a framework that disguises user browsers as anti-phishing crawlers when requesting web page content. When visiting web sites, Spartacus mutates the information that phishers may fingerprint in the HTTP request such as User-Agent, Referrer, or IP address [1] to make the request appear to be from an anti-phishing crawler. When the server-side cloaking script examines the HTTP request, it will decide that the visit is from an anti-phishing system, and will return a benign-looking web page to the user, sparing them from the phishing attack. Spartacus is quite effective at defending users against server-side cloaking, as I demonstrate in several evaluations. First, to estimate the potential benefits of Spartacus, I measured the prevalence of server-side fingerprinting-based cloaking techniques in phishing kits (programs used by phishers to easily create phishing web sites) using an automated analysis. In total, I analyzed 2,933 phishing kits and discovered that 96.52% (2,831) of them contain server-side fingerprinting-based cloaking techniques. Then, I performed an evaluation to see if Spartacus can trigger evasion in real-world phishing sites. In my large-scale evaluation of the framework, over a period of nine months from late-2020 to mid-2021, Spartacus visited 160,728 real phishing web sites (provided by the Anti-Phishing Work Group) *and evaded 82.28% of them without relying on blacklists or other anti-phishing techniques*. Because the Spartacus framework is designed to

[1] IP address is optional due to privacy concerns, as discussed in Section 5.3.

protect end-users directly, its impact on the functionality of benign web sites is a concern. I evaluated the performance and functionality impact of Spartacus on benign web sites both automatically and manually. I found that with Spartacus installed, the tested benign web sites displayed properly (i.e., benign web sites do not perform server-side cloaking or otherwise change the HTML that they send based on the mutated HTTP headers). When visiting benign web sites hosted on providers that employ security mechanisms such as Akamai and Cloudflare, Spartacus could successfully visit the majority of them (99.84% out of 10,000). Complex components in these web sites, such as buttons/links, online chat, register/login, shopping carts, checkout, etc. functioned correctly without any error. I also installed Spartacus in their daily-use browsers to visit web sites for one month, and Spartacus did not cause any issues when used in real-world web browsing for a month. I also evaluated current anti-phishing systems against modern phishing web sites and compared that to Spartacus. I submitted the 45,526 phishing web sites that Spartacus visited as part of my large-scale evaluation to anti-phishing systems and monitored the result. After waiting five days for blacklists to update, 24,154 (53.1%) phishing sites that were evaded by Spartacus (i.e., they showed benign content to Spartacus) *were not detected by anti-phishing systems*, 16,698 (36.7%) were evaded and detected, 4,598 (10.1%) were not evaded but were detected, and 76 (0.2%) were neither evaded nor detected. In other words, Spartacus alone can protect users against 89.8% of this subset of phishing sites I analyzed *in real time*, and the combination of Spartacus and current techniques can protect users against 99.8% of them, whereas existing techniques alone protect against only 46.8% (and these have a median blacklisting lag time of 2.58 hours, compared to Spartacus' real-time effect). These results suggest that the idea of Spartacus traps phishers in a dilemma: to attack Spartacus users, phishers should disable at least some server-side cloaking criteria and therefore allow more HTTP requests to successfully retrieve the phishing content. However, this strategy allows anti-phishing crawlers to view the phishing content and use content-based detection

techniques.

As I notice when proposing Spartacus, the anti-phishing ecosystem is passively protecting users from phishing websites. What it does is to analyze reported phishing URLs and detect/blacklist them. However, phishing attacks develop fast so they are often ahead of the anti-phishing ecosystem in the cat-and-mouse game. Any development of their phishing techniques may disable the current anti-phishing techniques. Similarly, in the IoT field, the secure world is a slave of the normal world and only does security operations that the normal world asks. Therefore, the secure world is passively protecting the IoT devices even with high privileges. Similar to the anti-phishing ecosystem, attackers who compromise the normal world can easily disable the secure world. So inspired by Spartacus, I propose iCORE, where the secure world is not a slave any more. Instead, it proactively monitors the operations in the secure world to identify any maliciousness. Like Spartacus, iCORE does not impact the benign activities in the normal world while monitoring it. iCORE is a novel continuous and proactive extrospection system with high visibility on IoT devices deploying multi-core ARM platforms exploiting ARM TrustZone extensions, to overcome the aforementioned limitations of current security tools. Dedicated cores named *Isolated Cores* are assigned to the secure world in the full power cycle starting from the system booting period to proactively, continuously, and stealthily extrospect the normal world. Secure boot procedure is deployed to help ensure the initialization of iCORE, which, in turn, provides a small trust computing base (TCB). The implementation of iCORE does not require any changes on the normal world operating system. Moreover, iCORE ameliorates the traditional master-normal-world and slave-secure-world concept by making the secure world play a role as a master of the system. Therefore, iCORE can execute its functionalities independently even if the normal world is compromised by an attacker. iCORE is designed as an integrity monitor with the attributes of continuousness, stealthiness, proaction, and high visibility. First, continuousness allows iCORE to detect any malicious modifications in time

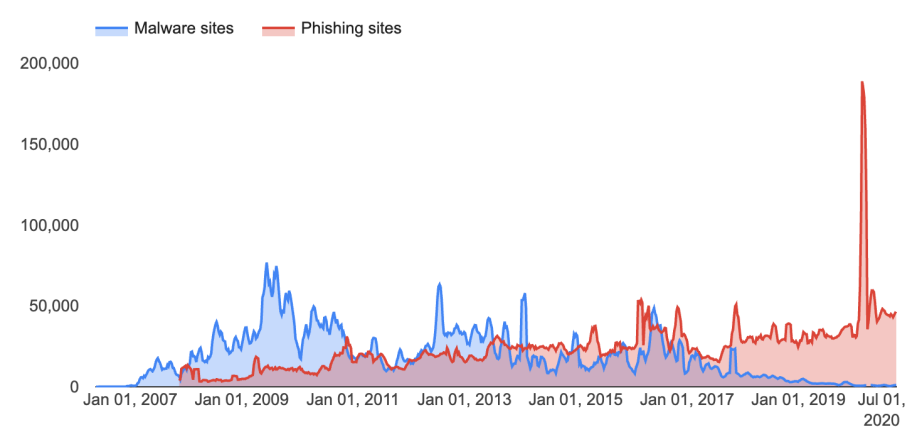


Figure 1.1: The Volume of Phishing Attacks Have Been Surging Year over Year [2].

when monitoring the normal world operating system. Event-driven monitoring methods, such as [30, 20, 21, 31], are easy to implement, but malicious operations cannot be detected or responded if certain events do not occur. Second, iCORE is staying and operating the monitoring functionalities in the secure world. Therefore, attackers in the normal world cannot detect its existence. Third, iCORE is out of the control of the normal world, hence it can provide proactive and independent monitoring functionalities over the normal world. Last, the high visibility is brought by the design of ARM TrustZone architecture that the secure world where iCORE resides can access all the resources of the normal world with the highest privilege.

Chapter 2

BACKGROUND

Over the past years, a myriad of techniques have been implemented by the anti-phishing ecosystem to detect and mitigate phishing attacks [4]. Analysis of phishing URLs [32, 33, 34, 35] and website content [36, 37, 38, 39, 40] has given rise to ecosystem-level defenses such as e-mail spam filters, malicious infrastructure detection, and URL blacklists.

Specifically, systems such as Google Safe Browsing [41] and Microsoft SmartScreen [42] power the anti-phishing backends that display prominent warnings across major web browsers when phishing is detected. These warnings are primarily blacklist-based: they rely on content-based detection. Evasion techniques commonly used by phishing websites are capable of bypassing or delaying such blacklisting [43, 5, 44].

2.1 Cloaking Techniques in Phishing

Attackers leverage *cloaking techniques* to evade detection by anti-phishing systems: phishing websites with cloaking display benign-looking content instead of the phishing page whenever they suspect that a visit originates from security infrastructure [4]. Server-side cloaking techniques identify users via information in HTTP requests [45]. Among them, *fingerprinting-based cloaking techniques* are widely used in advanced phishing web sites. Figure 2.1 depicts how phishing web sites use fingerprinting-based cloaking techniques. Cloaking code in the phishing web server fingerprints the profile in the HTTP request and responds with different web page content (with the goal of showing phishing content only to potential victims). Client-side cloaking is implemented through code that runs in the visitor's browser (JavaScript) to apply filters using attributes such as cookies or mouse movement.

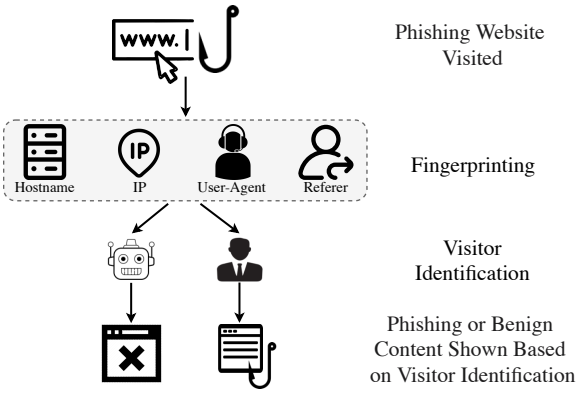


Figure 2.1: Typical Operation of Server-side Fingerprinting-based Cloaking in Phishing Web Sites.

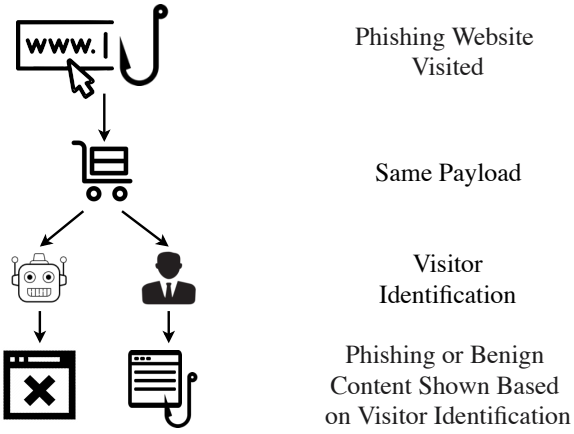


Figure 2.2: Typical Operation of Client-side Cloaking in Phishing Websites.

Existing anti-cloaking methodologies focus on bypassing server-side cloaking by comparing the visual and textual features of different versions of a crawled website retrieved by sending multiple web requests with different configurations (e.g., user agents or IP addresses) [46, 45, 7]. Client-side cloaking techniques, however, are still poorly understood due to challenges in automatically analyzing JavaScript code and understanding its semantics. Moreover, neither the prevalence nor impact of client-side cloaking has been investigated in the context of phishing.

Figure 2.2 shows how client-side cloaking techniques are used in phishing websites.

Cloaking code embedded in the HTTP response payload shows different web page content based on the identification of visitors (as either potential victims or bots). Consequently, cloaked phishing websites may have a longer life span than ones without: by delaying or avoiding detection, the attackers who launch these websites maximize their return-on-investment [5]. Because client-side evasion techniques enable complex interactions between potential victims and phishing websites, they may be more effective in hampering automated detection than traditional server-side cloaking, and, thus, pose a threat to potential victim users.

2.2 Challenges in Analyzing Client-side Cloaking

Unlike server-side code, the client-side code (JavaScript) of websites can trivially be obtained through crawling. Therefore, malicious websites typically leverage code *obfuscation* methods such as string array encoding, object key transformation, dead code injection, and even full encryption [47, 48]. Attackers also can dynamically generate and execute code (e.g., using `eval`) to hide malicious behaviors. Such obfuscation methods pose a challenge for static code analysis approaches, which are otherwise favored for their efficiency.

Other types of obfuscation also seek to prevent dynamic analysis approaches from detecting malicious behaviors. Malicious JavaScript code often targets specific versions of web browsers and operating systems by *fingerprinting* them [25]. Such attacks are difficult to discover because detection systems require extensive resources to reveal the conditions that trigger attacks [47]. Besides, external and inter-block *dependencies*, which require recording states in different execution paths, can be obstacles that thwart the analysis of JavaScript code [22]. Furthermore, scripts may execute in an *event-driven* manner to necessitate external triggers to initiate malicious behavior while otherwise appearing benign [22].

All of the aforementioned anti-analysis methods can potentially be leveraged by phishing websites' implementations of client-side cloaking techniques. Given the difficulty of analyzing such cloaking, the security community struggles to thoroughly understand the impact and prevalence of phishers' tactics, and, thus, may fail to appropriately mitigate them. When we consider the scale on which phishing attacks occur [49], the consequences of the corresponding gaps in detection and mitigation can be significant.

2.3 Redirection

Redirection refers to an activity that when a web browser attempts to visit a URL that has been redirected, a web page with a different URL is opened. The implementation of redirect can be through server (e.g., 3XX status code) or in the client browser (JavaScript and meta tag in HTML). Redirection contains both benign and malicious usage. People implement redirection to shorten long URL links, prevent broken links, protect privacy, and etc. However, miscreants (phishers) leverage such technique to evade detection from anti-phishing systems. Criminals often shorten the phishing URLs spread in social media so that ordinary users may not notice whether the link is legitimate [50]. They also exploit the open redirect vulnerabilities found in Google, YouTube, Google Docs, and etc., where a legitimate website allow any redirections, to bypass the URL based anti-phishing systems [51]. Last but not the least, phishers implement redirect chains, where more than one redirection occurs from the entry URL to the destination [52]. As researchers proposed several mechanisms to detect and mitigate malicious use of redirection in phishing, the miscreants began to embed advanced evasion techniques such as cloaking techniques to power the redirection.

2.4 IoT Platform and Security

2.4.1 Monitoring Mechanisms

In-the-box vs Out-of-the-box

To prevent and detect potential attacks on OS kernels, researchers developed two categories of approaches, namely ”in-the-box” approach and ”out-of-the-box” approach [15]. ”In-the-box” approach utilizes system software where the protection and monitoring tools reside in the monitored system. One instance is kGuard [53], which is a compiler plug-in to reinforce the kernel to detect return-to-user attacks. Such approaches have been proved less practical because the security tools can be disabled or even eliminated if the OS kernel is compromised [31]. Based on this observation, the research community has realized that the monitoring and protection tools should be correctly isolated from the monitored system so that attacks on the monitored system will not affect the security tools [20]. As a result, ”out-of-the-box” approach was proposed. ”Out-of-the-box” approaches, such as [30, 20, 21, 31], purport to deploy security tools outside of the monitored system. The ”out-of-the-box” approaches can be further categorized into hypervisor-assisted and hardware-assisted approaches based on the components that the security tools reside and take advantage of [31].

Hypervisor-assisted vs Hardware-assisted

The essence of hypervisor-assisted approaches is to utilize virtualization to provide security tools with a higher-privileged and isolated execution environment. Residing in a hypervisor or a virtual machine monitor (VMM), which runs on host’s hardware to control the hardware and guest operating systems, the security tools such as [54, 55, 56, 57, 58] can inspect the monitored operating system along with its interaction with hardware resources, and thus can detect potential attacks. However, the hypervisor itself is fragile because

it also contains large code base bringing countless vulnerabilities, such as [59, 60, 61]. Although sacrificing the whole hypervisor as a security tool can avoid the hypervisor being compromised, it is an unrealistic way because this idea will block the virtualization functionalities [21]. Furthermore, the high-cost of virtualization on IoT devices makes it difficult to deploy hypervisor-assisted protection approaches [20].

The hardware-assisted approach is also prevalent, where researchers develop their security tools with the help of isolated hardware. Some protection methodologies are designed to monitor kernel objects using external hardware [31, 30], which increase the cost of security tool deployment. Since 2004, new isolated hardware architectures, such as ARM TrustZone [16], AMD SVM [17], and Intel TXT [18], were proposed to provide a trusted area for secure executions. The essence of the isolated architectures is to physically segment the system resources into two worlds, named as the normal world (also as rich execution environment, REE) for conventional processing, and the secure world (also as trusted execution environment, TEE) for secure processing, respectively. This design guarantees that the security-sensitive data can be properly protected in the secure world [62]. With the assist of isolated hardware architectures, researchers have been transferring their focus to developing security tools based on the trusted execution environment. Among the security tools based on TEE, TZ-RKP [20] and SPROBES [21] are recent examples, which exploit the secure world to deploy the security tools. Both of the security tools protect the Linux kernel code area by trapping the normal world page table update operations and judging in the secure world if the operation is legitimate.

However, the execution time of workload increases due to performing world switch each time when a page table update occurs. In addition, modifications on both the normal and secure world OS before implementing the security tools are needed to interrupt the page table updates. Also, the attackers who have successfully compromised the normal world can opt out the functionalities of the secure world.

Event-driven vs Continuous

Lunt et al. [63] proposed a prototype real-time intrusion-detection expert system (IDES). In IDES, discrete and continuous measures were discussed. A discrete measure is used on a finite and unordered set of range of values, while a continuous measure is deployed for an infinite and continuous set of range of values.

Some of the modern monitoring and protection mechanisms implement a discrete variant, event-driven, aiming to detect after certain events happen. Vigilare [30], Ki-Mon [31], TZ-RKP [20], and SPROBES [20] use different techniques to implement their security tools, but they all leverage the concept of event-driven.

Other techniques, such as HookSafe [64] and SecVisor [65], aim to protect the kernel code continuously, which means the protection methods keep the integrity of the kernel not affected by events taking place.

2.4.2 ARM TrustZone Architecture

Overview of TrustZone

ARM TrustZone [16] is designed as a hardware-assisted security extension to ARM architecture, such as ARM Cortex-A and Cortex-M. ARM TrustZone physically partitions the system into two worlds, namely the normal world and the secure world. Each world has its own banked registers and memory which are running on the world-specific operating systems and applications. Trusted applications (TA) can execute secure processes in the secure world. Client applications (CA) are running in the normal world to operate conventional processes. Shared memory and general registers are used to communicate between the secure world and the normal world. Processes that are running in the normal world must call the `smc` [66] instruction to trigger one of the services in the secure world in order to request the security-sensitive data, and the secure world will send the data back to

the normal world if the request is accepted. The services that execute in the secure world, however, can access the resources in the normal world without permission from the normal world.

The secure world is proposed to help the normal world operate security-sensitive executions with high privileges to protect the secure data from being attacked and leaked. So it only executes the functionalities that the normal world requests. Hence, the secure world by the default design plays a role as a slave of the normal world even with the highest privileges among the whole system. Take comparing Apple Touch ID [67] as an example. The normal world calls `smc` instruction along with the fingerprint collected from the sensor to request a Touch ID comparison in the secure world. The secure world then executes certain functionalities and returns the result back to the normal world. However, it cannot invoke such operation without the request of the normal world. From this perspective, the functionalities of the secure world will not be executed if the normal world never calls them.

Core Initialization with Secure Boot

ARM Trusted Firmware (ARM-TF) is one of the major booters that is designed to initialize the ARM cores with TrustZone extension. Both cold boot, where the system is switched on physically, and warm boot, where cores have already been initialized, should go to ARM-TF reset entry point. Afterwards, different booting procedures step into their own initialization sequence.

For the cold boot, the initialization of hardware, including core, platform, and architecture, is performed first. The primary core, which is selected when it is released from reset and executes mainly the cold boot path, starts with the C runtime initialization. And the other cores, called secondary cores, are placed in a platform-specific state and wait to be woken up after the primary core finishes initializing enough functionalities [68].

As for the warm boot procedure, the system goes to the warm boot entry point to continue the configuring PSCI, platform, architectural, and generic setup, along with PSCI state maintenance [68].

During the initialization, secure boot procedure is exploited to protect the integrity of all the secure world software images from being unauthorized or illegally modified, applying cryptographic checks to every stage of the secure world booting procedure [69]. For example, a trusted vendor would sign the image that she plans to execute on the device with her private key and then send the image along with the signature to this device. The corresponding public key is stored and protected from being substituted to verify whether the image has been tampered with and whether it is from the trusted vendor. The secure boot procedure also exploits the concept of chain of trust, meaning that starting from the root of trust that located in on-SoC ROM, every other software component can be verified by its higher level component before being executed.

Normal World Memory Access from Secure World

User and kernel processes running in the normal world have their own private virtual address memory space, which is the contribution of MMU. When a process in the normal world wants to access the memory through the virtual address, MMU will convert the virtual address through translation tables to the corresponding physical address to access the memory.

Although the secure world can access all the resources in the normal world, physical addresses are required by the secure world to access the specific normal world memory. Since there is no function designed by default to convert the virtual address to the physical address, we design one which will be discussed in detail in Section 6.2. After acquiring the physical address, the secure world can access the static kernel memory in the normal world directly. In addition, the static kernel data is linearly mapped in the normal world memory.

Therefore, with the starting and ending physical addresses, the secure world can load the whole data stored in the static memory area [70].

Chapter 3

UNDERSTANDING AND CATEGORIZING SOPHISTICATIONS IN PHISHING

As mentioned above, sophisticated phishing websites contain different evasion techniques to circumvent anti-phishing systems, such as cloaking and redirection. So in this chapter, I first present CrawlPhish to mitigate sophisticated phishing attacks from the perspective of detection.

Client-side cloaking techniques can help phishing websites evade detection by anti-phishing entities [5], yet prior studies have not investigated them in detail, despite evidence that sophisticated phishing websites—such as those with client-side cloaking—are responsible for a majority of real-world damage due to phishing [71].

I discover eight different types of JavaScript cloaking techniques across three high-level categories: User Interaction, Fingerprinting, and Bot Behavior (summarized in Table 3.1). Cloaking techniques in the *User Interaction* category show phishing content only if visitors interact with a phishing website (e.g., by moving the mouse or clicking a specific button). Phishing websites with *Fingerprinting* identify visitors by inspecting the configuration of browsers or web requests. Finally, phishing websites with *Bot Detection* identify anti-phishing crawlers based on factors such as how long the web page stays open and whether the web request is repeated after failing initially.

I aim to comprehensively understand and characterize the landscape of client-side cloaking techniques used by phishing websites in the wild through an automated methodology for analyzing them. To this end, I design, implement, and evaluate *CrawlPhish*: a framework that automatically detects and analyzes client-side cloaking within phishing websites. Figure 3.1 provides an overview of the CrawlPhish architecture. CrawlPhish is composed of the following components:

Cloaking Category	Cloaking type	Requirement
User Interaction	Pop-up	Click on alert/notification window
	Mouse Detection	Move mouse over browser
	Click Through	Pass Click Through on browser
Fingerprinting	Cookie	Check document.cookie
	Referrer	Check document.referrer
	User-Agent	Check navigator.userAgent
Bot Behavior	Timing	Render webpage after certain time using sleep()/Date.getTime()
	Randomization	Show content randomly using Math.random()

Table 3.1: Summary of the Client-side Cloaking Technique Types Identified in This Work.

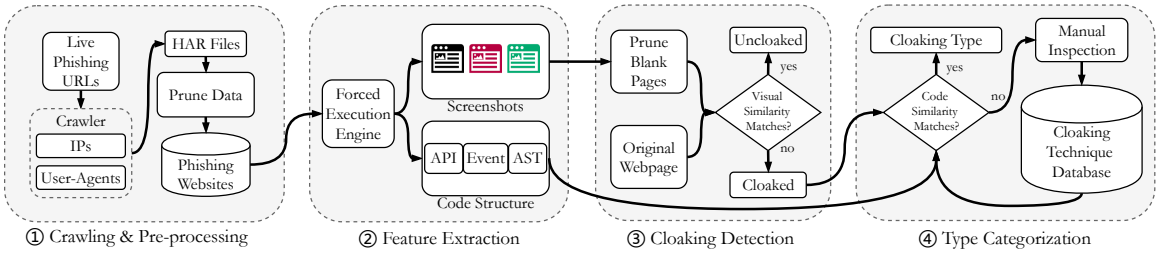


Figure 3.1: CrawlPhish Architecture.

① **Crawling and pre-processing (§3.1):** CrawlPhish first collects web page source code (along with any external file inclusions) by visiting live phishing website URLs recently reported to anti-phishing feeds. We then filter URLs that cannot be retrieved as well as URLs without any JavaScript code.

② **Feature extraction (§3.2):** CrawlPhish adapts a state-of-the-art code analysis method, *forced execution* [22], to execute JavaScript regardless of branch conditions, and extracts all possible execution paths in which evasion techniques could be implemented. We then derive (1) visual features of the rendered web pages, by means of screenshots, and (2) code structure features such as web API calls, event listeners, and the Abstract Syntax Tree (AST) for each path.

③ **Cloaking detection (§3.3):** CrawlPhish analyzes the visual features corresponding to

each execution path to detect if cloaking exists, and it stores the corresponding code structure features of every such path.

④ **Cloaking categorization (§3.4):** Using the code structure features, CrawlPhish categorizes the cloaking techniques used by phishing websites based on their semantics.

3.1 Crawling & Pre-processing

To collect the source code of *live* phishing websites to detect and classify client-side evasion methods that are currently employed in the wild, CrawlPhish first obtains URLs of known phishing websites in real-time.

In our deployment, CrawlPhish continuously ingested URLs from the APWG eCrime Exchange database—a curated clearinghouse of phishing URLs maintained by various organizations engaged in anti-phishing. Because this database receives frequent updates and tracks phishing URLs that target a diverse range of brands, it is well-suited for phishing website analysis. [1] Note, however, that the inclusion of a URL in the database does not mean that it was adequately mitigated (e.g., through timely blacklisting) [44]. Hence, websites found to use sophisticated client-side cloaking still warrant scrutiny.

Next, CrawlPhish downloads source code by visiting each phishing website URL (shortly after being ingested) using a programmatically controlled web browser. Specifically, CrawlPhish stores source code using HAR files, which capture all HTTP requests/responses between our client and the server, and ensure that all dependencies (such as linked scripts) are preserved for each website. In case of a failed request, CrawlPhish switches between different configurations of IP addresses and user-agents in an effort to circumvent potential server-side cloaking techniques used by phishing websites [4]. 4,823 of the 128,973 web-

[1] Although the goal of cloaking is to evade detection by automated anti-phishing systems, such evasion will often delay detection rather than outright prevent it. Phishing websites may also be detected by other means (e.g., manual review) [71]. Thus, we expected the AWPG database to contain a representative sampling of any client-side cloaking that might be used in the wild.

sites we crawled (3.74%) showed different response status codes after we switched request configurations.

Finally, CrawlPhish filters out URLs that contain blank pages or non-phishing websites. Such websites were either already taken down [72] or were false-positive detections by the time of crawling. We found 0.53% of URLs within the APWG Dataset to be false positives. Therefore, CrawlPhish excludes data in the following cases:

i. *empty* websites: servers respond with no content.

ii. *error* websites: requests for URLs were denied because the phishing websites were already taken down, or used server-side cloaking which we could not bypass.

iii. *non-phishing* websites: mistakenly reported URLs, which CrawlPhish filters based on a manually curated whitelist of reputable domains.

3.2 Feature Extraction

Cloaked content detection. Client-side cloaking techniques used in phishing websites can be more diverse than server-side cloaking because they can not only fingerprint visitors based on configurations of browsers and systems, but may also require visitors to interact with websites. To effectively detect client-side cloaking techniques, CrawlPhish adapts J-Force: a *forced execution* framework implemented in the WebKitGTK+ browser that executes JavaScript code along all possible paths, crash-free, regardless of the possible branch conditions, event handlers, and exceptions [22]. We modified J-Force to whitelist (avoid force-executing) well-known JavaScript libraries, such as *Google Analytics* or *jQuery*, to expedite execution by ignoring the benign content changes that such libraries could introduce.

Execution time limit. We select a time limit for each invocation of forced execution by CrawlPhish to avoid failures due to long-running scrips (e.g., due to heavy branching or

long-running loops).

As a starting point, we chose an execution limit of 300 seconds. We conducted an experiment by force-executing 2,000 randomly selected phishing websites in our crawled dataset to record the execution time. We found that 1.75% of phishing websites contained JavaScript code that exceeded the time limit. Execution finished as quickly as 12.56 seconds, the median execution time was 13.82 seconds, the average execution time was 29.96 seconds, and the standard deviation was 54.89 seconds. Based on this experiment, we chose a final execution limit of 195 seconds (three standard deviations above the mean) so that CrawlPhish could efficiently analyze the majority of phishing websites.

Feature extraction. To facilitate detection of (the existence of) cloaking and categorization of the corresponding cloaking type, CrawlPhish extracts both *visual* and *code structure* features from each phishing website. Each phishing website's visual features consist of the set of all web page screenshots (in our implementation, at a resolution of $2{,}495 \times 1{,}576$ pixels) captured after every possible execution path is explored by forced execution. In our dataset, each website generated 46.3 screenshots on average. CrawlPhish compares the screenshots of *each* execution path within one website against the original screenshot to detect if cloaking exists, because the presence of cloaking will result in significant visual layout changes [45]. The code structure features include web API calls, web event listeners, and ASTs, which can characterize different types of cloaking techniques and reveal how the cloaking techniques are implemented. Using forced execution, CrawlPhish can reveal and extract the web APIs and events contained in every code block, even if the code is obfuscated. CrawlPhish can then classify the cloaking types in a website using the code structure features.

Code structure features used. According to preliminary analysis which we conducted by manually inspecting cloaking techniques in a sampling of phishing websites in our dataset, different client-side cloaking techniques each have substantially different features. For

example, a cloaking technique that checks mouse movement waits for an `onmousemove` event, then performs DOM substitution or redirection. However, a cloaking technique that checks screen size would first access the `screen.height` property. Therefore, as CrawlPhish executes a code block via forced execution, it records the web APIs and events that are invoked in the code block.

In addition, we found that the same semantic types of client-side cloaking techniques have many different implementations. CrawlPhish distinguishes between different implementations of each type of cloaking technique by comparing ASTs. Even though JavaScript code is often obfuscated, the AST feature is still useful because most phishing websites are deployed using phishing kits, so the corresponding websites, with the same phishing kit origin, share the same source code structure [6]. Furthermore, by computing the AST similarity, we can trace the origin of the cloaking technique by finding similar implementations earlier in phishing pages.

3.3 Cloaking Detection

CrawlPhish examines the *visual similarity* between force-executed screenshots and a screenshot of the website rendered in an unmodified version of WebKitGTK+ (i.e., as would be shown during a normal browser visit) to detect if cloaking exists. Because phishers implement JavaScript cloaking techniques to *evade* detection by anti-phishing systems, they remove suspicious attributes in websites (e.g., login forms) or outright redirect to a benign website. Therefore, the visual content shown when the cloaking condition is not satisfied will differ significantly from that of the malicious page.

For example, consider a phishing website that asks visitors to click on a button in a pop-up window prior to showing the phishing content. After forced execution, two different execution paths result in two different screenshots: one as an initial benign-looking page (Figure 3.3a), and the other with phishing content (Figure 3.3b). Therefore, we con-

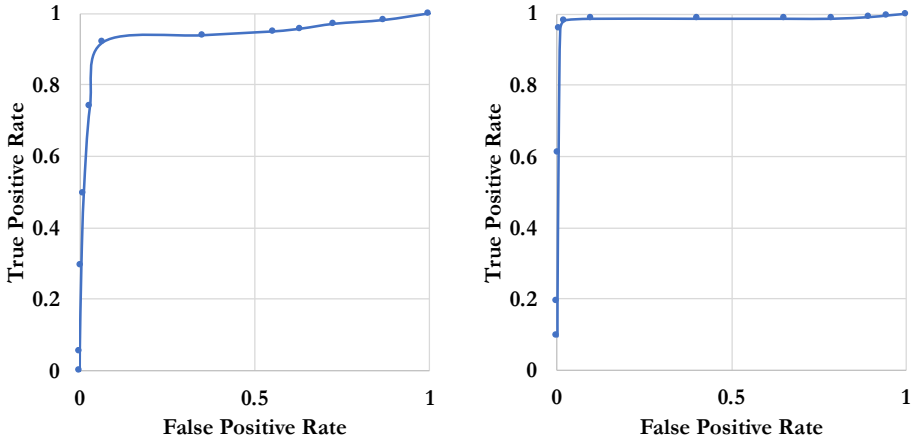
sider a phishing website as *cloaked* if any of the screenshots taken during forced execution noticeably differ from the original one.

CrawlPhish can also reveal phishing content hidden behind multiple layers of cloaking. Consider a phishing website with a cloaking technique that (1) detects mouse movement and (2) checks the referrer such that the malicious content will appear only if both requirements are met. CrawlPhish will explore the execution path that shows the malicious content by force-executing it, regardless of the branching conditions. Therefore, after each screenshot is compared with the screenshot of the original page, CrawlPhish determines that a cloaking technique exists because one of the screenshots will differ.

Removal of blank pages after forced execution. Screenshots of pages rendered by force-executed paths may be blank, which can be caused by (1) negative branches from cloaking techniques (such as mouse movement detection) that require user input or (2) execution paths that take longer to finish than the execution time limit. In the latter case, CrawlPhish can mislabel a website as *cloaked* if an initial screenshot is compared to an empty page caused by unfinished execution paths. For example, phishers may trigger an infinite loop if they identify that a visit is from an anti-phishing system. In this case, CrawlPhish cannot finish forced execution and hence the screenshot remains empty. Thus, a current limitation of CrawlPhish is that it cannot detect cloaked websites with very long execution times. However, according to our evaluation, this situation does not happen often: only in 1.75% of the websites we considered.

Detection algorithm. To perform visual similarity checks between the screenshots, we implement the pHash algorithm [73], which compares visual similarity with robustness and good discrimination. We calculate pHash scores between the original screenshot and those captured after each path finishes execution.

$$score = pHash(S_{original}, S_i), i \in [1, 2, ..., n] \tag{3.1}$$



(a) Visual feature threshold.

(b) Code structure feature threshold.

Figure 3.2: ROC Curves to Select Thresholds for Cloaking Detection and Cloaking Type Categorization.

In Formula 1, S represents each screenshot and n is the number of screenshots captured from forced execution. We consider two screenshots to be similar (no cloaking) if the pHash score is less than a threshold (5.0) that we set based on preliminary testing results on 1,000 phishing websites. Differing screenshots will have a score of 5.0 or greater. Figure 3.2a shows the ROC curve for selecting the visual similarity threshold. We selected the threshold that provides a 92.00% true-positive rate with a 6.77% false-positive rate. We note that our evaluation in Section 5.4 shows that CrawlPhish exhibited higher detection accuracy (98.25%) with a lower false-positive rate of 1.45% than what was indicated by the threshold in the ROC curve.

3.4 Cloaking Categorization

Once CrawlPhish detects the *presence* of cloaking on a web page, categorization of the specific *type* of cloaking allows us to measure and understand the prevalence of different high-level client-side cloaking techniques used by phishers. To facilitate this categoriza-

tion, CrawlPhish maintains a *cloaking technique database* that contains the code structure features for each instance of cloaking, annotated with the corresponding cloaking semantics. Using the database, CrawlPhish can not only identify known cloaking types, but also provide detailed information about emerging cloaking techniques.

Initial database. We first obtained 1,000 cloaked phishing websites (true positives), for which we used CrawlPhish to determine the existence of client-side cloaking. Then, we manually examined the source code of the phishing websites to label the corresponding cloaking techniques. We also recorded code structure features as ground truth.

For example, we labeled one type of cloaking technique as *Mouse Detection* if the recorded code features have the `onmousemove` event and use the `location.href` API. Over time, as CrawlPhish executes, if the presence of cloaking is detected on a website but the code features *do not* sufficiently closely match any of the records in the database, the website is flagged for manual review such that the missing features (and, potentially, new cloaking types) can be populated. Otherwise, the website is automatically labeled with the corresponding semantic cloaking type. Within the dataset we crawled, manual effort was rarely needed after we populated the initial database. Thus, this requirement does not impede the automated operation of our framework.

Categorization algorithm. CrawlPhish employs the Hamming Distance (HD) algorithm [74] to compute the similarity of the API calls and web events. To this end, we use an array data structure with one position for each of the 4,012 types of web API calls or events as defined by the Mozilla MDN [75, 76], which documents currently available web APIs. At each position in the array, we store the number of corresponding API calls or events as observed by CrawlPhish. We then convert this array to a fixed-length string (e.g., string[0] is the number of `ActiveXObject` in the code block and string[1] stores the amount of `Date` API calls) so that we can apply the HD algorithm. Thus, the result of the HD algorithm on a pair of strings represents the similarity of web APIs and events between

two code blocks. Lower HD values indicate higher similarity.

We also leverage JSInspect [77] to find structurally similar code snippets based on the AST. This will identify code with a similar structure based on the AST node types (e.g., BlockStatement, VariableDeclaration, and ObjectExpression). We combine these approaches to overcome limitations of code similarity checkers based solely on either ASTs or API calls. Consequently, by comparing the code structure similarity of all suspicious code blocks against records in the database, all known cloaking types can be identified in one website (even if there are multiple types). If the features of a suspicious code block are not sufficiently similar to any record in the database, we will manually examine it, label the cloaking type, and then add it to the database, which is the only process that requires manual effort in the CrawlPhish framework.

Similar to the visual similarity check, we empirically set a threshold for the code similarity check based on preliminary manual analysis of 1,000 cloaked phishing websites. We consider only two categories to find a threshold: correctly labeled cloaking types and mislabeled cloaking types. Per Figure 3.2b, we selected a code structure threshold with a true-positive rate of 95.83% and a false-positive rate of 0.79%. When CrawlPhish compares the code structure features of a new phishing website to ones in our database, the AST similarity score must be greater than 0.74 *and* the Hamming Distance of web APIs and events must be within 34 for a new website to be marked with a known type of cloaking technique.

3.5 Evaluation: Detection of Cloaked Phishing Websites

In this section, we evaluate the client-side cloaking detection accuracy of CrawlPhish. In this experiment, we first randomly sampled and manually labeled 2,000 phishing websites that did not contain JavaScript cloaking techniques as well as 2,000 phishing websites with various types of client-side cloaking. We then ran CrawlPhish to detect if client-side

Crawled Phishing Websites From APWG		Analyzed	
		Cloaked	**Non-cloaked**
Actual	**Cloaked**	TP 1,965 98.25%	FN 35 1.75%
	Non-cloaked	FP 29 1.45%	TN 1,971 98.55%

Table 3.2: Accuracy of Cloaking Detection by CrawlPhish.

cloaking exists. Finally, we compared the automated cloaking detection results against our manually labeled ground truth dataset to calculate the detection accuracy.

Table 3.2 shows the confusion matrix of CrawlPhish's detections. Within the 4,000 phishing websites, CrawlPhish correctly detected 1,965 phishing websites as cloaked and 1,971 as uncloaked, with a false-negative rate of 1.75% (35) and a false-positive rate of 1.45% (29). Note that unlike a general phishing detection tool that should prioritize false positives over false negatives [41], the client-side cloaking detection component in CrawlPhish does *not* critically need to do so, because the goal of our detection is to study the nature of client-side cloaking, rather than to detect a phishing attack. If CrawlPhish trades higher false negatives for lower or even zero false positives, the study might be less complete because we might miss many relevant instances of cloaking. Therefore, the detection of CrawlPhish should balance false positives with false negatives.

Each of the 29 false-positive cases was caused by one of two errors. The first error was due to the rending overhead of the unmodified browser which loaded the original phishing page. WebKitGTK+, the web browser we used in the CrawlPhish framework, failed to render the original websites within an allotted time limit due to a large number of CSS and JavaScript files included by the website. As a result, the original screenshot of each website was blank, but the screenshots after forced execution were not blank, so CrawlPhish mislabeled the corresponding websites as *cloaked* because the screenshots differed before and after forced execution. The second error was caused by inaccuracies in our image

similarity checks. The image similarity check module erroneously distinguished between screenshots of identical pages due to slight variations in the page layout generated by the browser with and without forced execution.

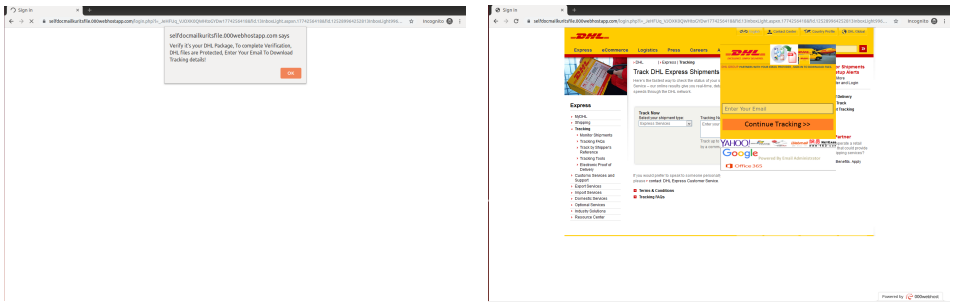
In terms of the false negatives, we found that 32 out of the 35 stemmed from a long execution time of cloaked phishing websites (similar to the first reason for false positives). Forced executed screenshots are not taken if an execution path takes too long to finish execution. We used a 195-second execution time window for each execution path. However, the paths that CrawlPhish does not execute due to a timeout may contain cloaking technique implementations. Without those screenshots, CrawlPhish cannot detect the cloaking technique, so it mislabels the corresponding website as *uncloaked*.

In three rare cases, real phishing websites appeared nearly blank due to low page contrast. For example, if phishing websites have a white background with light text, CrawlPhish would not distinguish between the corresponding screenshot and a blank one. We manually examined these cases and found that CSS inclusions were missing from those websites (i.e., they could not be retrieved by our crawler).

Client-side cloaking occurrence statistics. Within our dataset of 112,005 phishing websites, CrawlPhish found that 35,067 (31.31%) phishing websites implement client-side cloaking techniques in total: 23.32% (6,024) in 2018 and 33.70% (29,043) in 2019. We note that cloaking implementations in phishing grew significantly in 2019. We hypothesize that phishers are either leveraging such cloaking because it increases their profitability or because improving detection systems make advanced evasion necessary, or both.

3.6 Evaluation: Cloaking Categorization

In this section, we elaborate on the eight types of client-side cloaking techniques detected by CrawlPhish (as previously introduced in Table 3.1). We also evaluate the accuracy of CrawlPhish's semantic cloaking categorization, track trends in the deployment



(a) Initial appearance. (b) Force-executed appearance.

Figure 3.3: Initial and Force-executed Appearance of a Phishing Website With *Pop-up* Cloaking.

and evolution of different implementations of these cloaking techniques, and analyze how frequently they are used.

3.6.1 Categorization of Cloaking Types

User Interaction: *Pop-up*. With this technique, phishing content remains hidden until a button in a pop-up window is clicked. Specifically, JavaScript code listens for an `onclick` event to evade anti-phishing bots. Figure 3.3 shows an example of a phishing website that implements this technique. The website in Figure 3.3a initially shows an `alert` window to an anti-phishing bot or a real user. Thus, this phishing website seeks to evade detection by anti-phishing bots because no phishing content or typical attributes (such as a login form or logos of a legitimate organization) are found on the page. However, CrawlPhish reveals the phishing content hidden behind the popup window as shown in Figure 3.3b.

Figure 3.4 shows a more advanced version of the pop-up cloaking techniques that CrawlPhish detected. Because an `alert` window can easily be closed through common browser automation frameworks such as Selenium [78] or Katalon [79], some phishers instead use the *Web Notification API* [80]. We observed that due to technical limitations, top automation frameworks [81] do not currently support interaction with web notifications.

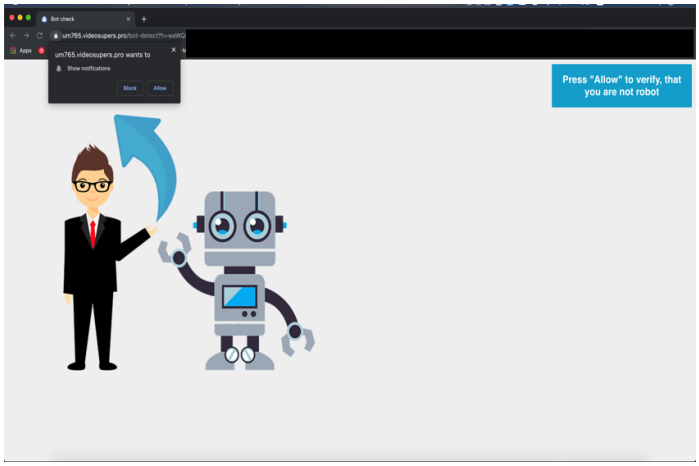


Figure 3.4: A Phishing Website with the Evolved Pop-up (Notification) Cloaking Technique, in Which the Web Page Directs Human Visitors to Click on the "allow" Button by Showing an Arrow.

These automated browsers opt to disable the notification window to avoid such interactions. Phishers, however, only allow visitors who actually click the "Allow" button to access the phishing content. Therefore, because the phishing website will not show any phishing content until a visitor clicks the "Allow" button in the notification window, it will evade detection. Phishers use a deceptive web page that asks visitors to click the button on the notification window, as shown in Figure 3.4. As an added benefit to attackers, by using a notification window, cloaked phishing websites could also directly send spam to visitors through their browsers (we do not evaluate the extent of such abuse). Through this, we show that criminals are using cutting-edge browser features to evade existing detection systems.

User Interaction: *Mouse Detection*. This cloaking type seeks to identify whether a website visitor is a person or an anti-phishing bot by waiting for mouse movement before displaying the phishing content. Specifically, the cloaking code listens for the `onmousemove`, `onmouseenter`, or `onmouseleave` events. This technique is used frequently by phish-

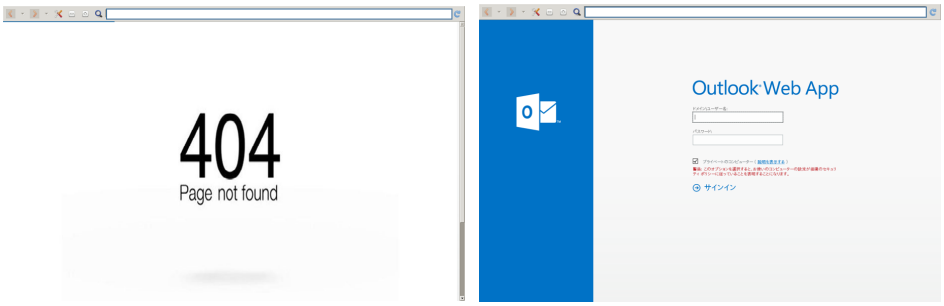
Cloaking Technique		Public Dataset			APWG Dataset					Identical Groups
		Unique Groups	Top Group		Unique Groups	Top Group		Earliest Impl.	Groups Used From 2018	
Category	Type		Count	Percentage		Count	Percentage			
Fingerprinting	Cookie	43	437	15.01%	28	325	7.39%	09/2018	12	14
	Referrer	27	156	5.85%	37	92	3.92%	08/2018	21	9
	User-Agent	65	563	53.31%	33	181	12.97%	07/2018	24	20
User Interaction	Pop-up: Alert	424	249	3.26%	335	73	1.21%	06/2018	276	127
	Pop-up: Notification	29	52	4.22%	17	284	18.67%	11/2018	7	11
	Click Through	105	1,541	22.88%	51	1,275	16.45%	10/2018	13	31
	Mouse Detection	87	138	6.81%	108	500	8.63%	06/2018	47	37
Bot Behavior	Randomization	73	42	16.03%	125	58	3.57%	09/2018	62	43
	Timing	597	387	7.76%	394	416	5.99%	06/2018	303	197

Table 3.3: Overview of the Number of Distinct Groups of Cloaking Code Implementations in the APWG and Public Datasets.

ers, and accounts for 16.53% of all cloaking technique implementations in Table 3.4, because most people have a habit of moving the mouse while a website is rendering in the browser [82].

User Interaction: *Click Through*. Some phishing websites require visitors to click on a specific location on the page before displaying phishing content [83]. Simple variants of this cloaking technique require visitors to click on a button on the page and are, thus, similar to `alert` cloaking. However, more sophisticated variants display fake CAPTCHAs that closely mimic the look and feel of Google's reCAPTCHA [84]. Given the common use of reCAPTCHA by legitimate websites, phishing websites with fake CAPTCHAs make it difficult for potential victims to identify that they are fake. If anti-phishing systems cannot access phishing content because of the Click Through technique, they may fail to mark the websites as phishing.

Bot Behavior: *Timing*. Some phishing websites show phishing content only at a certain time, or deliberately make rendering slow by using the `setTimeout()` or `Date.getTime()` APIs. If phishing websites take a longer time to render than thresholds set by detection systems, such websites can evade detection. Actual visitors, however, might wait for the completion of web page rendering [85].



(a) Benign page shown when cookies are disabled.

(b) Force-executed version, which reveals the login form.

Figure 3.5: Appearance of a Phishing Website with The *Cookie* Cloaking Technique.

Bot Behavior: *Randomization*. Some phishers try to evade detection by using a non-deterministic mechanism: such phishing websites generate a random number before the page is rendered, and only show phishing content if a certain threshold is met. Anti-phishing crawlers or human inspectors may not visit the same website again if it initially shows benign content. Therefore, this technique may appear to be a "dumb" way to evade detection by anti-phishing systems. However, its use in the wild suggests that it may be worthwhile: we suspect that phishers who use this technique are aware of the conditions for detection by anti-phishing entities and try to trick anti-phishing bots with a non-deterministic approach to cloaking.

Fingerprinting: *Cookie*. Similar to server-side cloaking techniques, client-side cloaking techniques can also check visitors' request attributes to fingerprint them. Figure 3.5 illustrates a phishing website that fingerprints whether a visitor is a person or an anti-phishing bot by checking if cookies are disabled in the browser. When cookies are disabled, the phishing websites will display benign content, as shown in Figure 3.5a. Some anti-phishing crawlers disable cookies to avoid being bound to a single session. However, CrawlPhish detects cloaked phishing content as shown in Figure 3.5b. Similarly, this cloaking technique may also test if the *browser cache* is enabled [86].

Fingerprinting: *Referrer*. Phishing websites can check whether incoming traffic originates from phishers' lures or other unwanted sources. Therefore, some phishing websites display benign content to visitors with a blank *Referer* [87], which could indicate that a URL was directly typed in. Similarly, referrals from search engines or known security domains can be blocked.

Fingerprinting: *User-agent*. Some phishing websites seek to identify anti-phishing crawlers based on their user-agent strings. The `navigator.userAgent` property stores information about the browser and operating system (e.g., Mozilla/5.0 (X11; Linux x86_64)). Therefore, anti-phishing bots such as *Googlebot* can be blocked as their `userAgent` property is a known value.

Combinations of cloaking techniques. Multiple client-side cloaking techniques are occasionally used together by phishing websites, as doing so may further increase evasiveness. For example, CrawlPhish found 503 instances of *Click Through* and *Referrer* used together. Also, we found *Timing* and *Cookie* in 476 cloaked phishing websites.

3.6.2 Accuracy of Cloaking Categorization

To evaluate the accuracy of CrawlPhish's categorization of cloaking types, we selected the same 2,000 cloaked phishing websites as in Section 5.4 (this set contains all three categories of client-side cloaking techniques) and manually labeled the correct cloaking type based on their code structure features. We, then, sent these websites through the feature extraction (②) and the cloaking detection (③) phases of CrawlPhish to locate the code blocks in which each cloaking technique is implemented. CrawlPhish checked the code structure feature similarity as populated over the course of our deployment (④). As stated in Section 3.4, CrawlPhish compares the code structure features of all snippets flagged by Step ③ with the records in the database to discover all possible cloaking techniques in a given phishing website.

We found that CrawlPhish correctly categorized the cloaking type with 100% accuracy. This high accuracy stems in part from the manual inspection involved when the code structure features of the examined snippet do not match any existing records in the database, as discussed in Section 3.4. Thus, we conclude that web API calls, web events, and ASTs suffice for distinguishing between different cloaking types, even when the underlying implementations vary.

3.6.3 Grouping of Implementations

Because phishing kits directly enable the scalability of phishing attacks and are readily available through underground markets [88, 6, 89], tracking the deployment and evolution of kits can help researchers and investigators pinpoint the threat actor (i.e., a kit author or criminal group) behind a series of phishing websites and identify the prevalence phishing attacks attributable to the same author. The web page source code collected by CrawlPhish is suitable for this purpose because such source code can be obtained for virtually any phishing URL—unlike server-side code [4].

By comparing code similarity between JavaScript snippets used by cloaked phishing websites, over time, we can group related cloaking technique implementations (i.e., implementations attributable to the same origin) together. Specifically, we compare the AST similarity among cloaking technique implementation code blocks to find matches using JSInspect [77] (the same technique we leveraged to check the code structure similarity). In Table 3.3, we provide an overview of the number of implementation groups that we found for each cloaking technique within the APWG Dataset and the Public Dataset. In addition, we compare the overlap in groups between the two datasets, and we determine the earliest date that each technique was observed.

Implementation groups in the APWG Dataset. We found that the earliest implementation of *each* cloaking type was in 2018. Also, we found that 1,128 groups account for

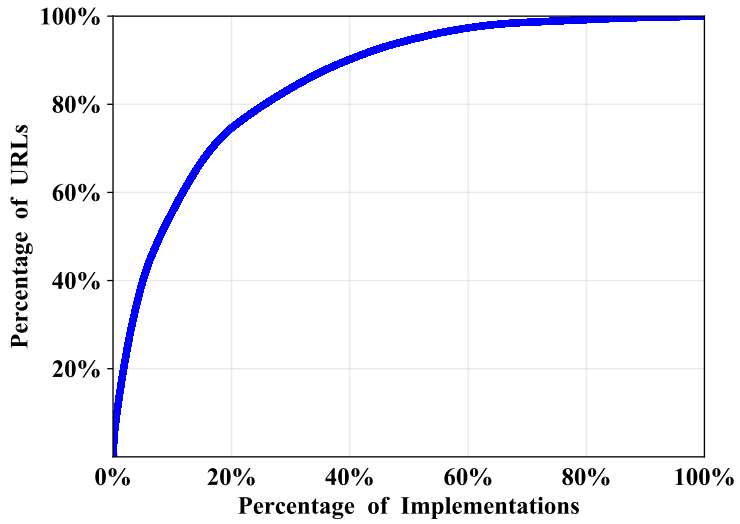


Figure 3.6: CDF of Implementation Groups for All Phishing Websites in the Apwg Dataset.

35,067 cloaked phishing websites detected by CrawlPhish. Figure 3.6 shows the cumulative distribution function (CDF) of unique implementation groups in the APWG Dataset: 20% of unique cloaking implementation groups account for 74.65% of all phishing websites. This shows that a small number of phishing kits is likely responsible for a significant proportion of sophisticated phishing websites in the wild. We discover that the Timing cloaking type has the most groups (394) among all cloaking types. Because this cloaking technique is less popular according to our findings, we suspect that prominent phishing kit developers do not deploy it, though individual criminals may still want to leverage it. Among the largest groups, we observe that one group of Click Through cloaking accounted for 16.45% (1,275) of code variants. As many as 18.67% (284) of the Notification Window occurrences were within a single group.

Implementation groups in the Public Dataset. We also compare the cloaking groups within the Public Dataset [90], which was sourced from OpenPhish [91], PhishTank [92], PhishStats [93], and other phishing URL archives. Using this additional dataset, we can validate that the APWG dataset was representative of the ecosystem and evaluate the exis-

tence of other cloaking techniques that may not have been present in the APWG dataset. Table 3.3 shows detailed statistics on the cloaking group distributions between the two datasets. The number of groups found for each cloaking type from both datasets is similar. The Timing and Alert cloaking techniques still have the highest overall number of groups, which matches the findings from the APWG dataset. The number of groups for Click Through cloaking, however, increases from 51 to 105. We suspect that different phishers are developing more phishing kits with this cloaking technique because they realize that it can effectively evade detection by anti-phishing systems.

In addition, by comparing the AST similarity of implementation groups between the Public Dataset and the APWG Dataset, we discover that the same groups of cloaking technique types exist in both datasets. 11 out of 17 distinct groups of the Notification cloaking technique in the APWG Dataset also appear in the Public Dataset. Additionally, the Alert and Timing cloaking techniques have the most identical groups between the two datasets. This result indicates that phishing kits leveraging client-site cloaking techniques are widely used.

Evolution of cloaking groups over time. Because we crawled phishing data in both 2018 and 2019 from APWG feed, this dataset enables us to trace the origin of each cloaking type. The Timing, Alert, and Mouse Detection cloaking techniques were first used in phishing websites from June 2018 in our dataset. The (more advanced) Notification technique first appeared in November 2018. The early occurrence of these evasion methods reminds us that phishers are trying to stay one step ahead of the anti-phishing ecosystem. While researchers and anti-phishing entities were working on mitigations against server-side cloaking techniques [7, 4], those attackers had already turned their focus toward implementing client-side evasion methods. We suspect that those client-side cloaking techniques may have already been employed well before June 2018 [7, 22] (the date we started crawling).

We also observe the evolution of cloaking techniques from the perspective of obfus-

Cloaking Technique			2018	2019	Total	Share
Category	Type		Count (%)	Count (%)	Count (%)	Count (%)
Fingerprinting	Cookie		1,295 (21.50%)	6,842 (23.56%)	8,137 (23.20%)	4,395 (12.53%)
	Referrer					2,346 (6.69%)
	User-Agent					1,396 (3.98%)
User Interaction	Pop-up	Alert	2,416 (40.11%)	17,782 (61.23%)	20,198 (57.60%)	6,027 (17.19%)
		Notification				1,521 (4.34%)
	Click Through					7,753 (22.11%)
	Mouse Detection					5,797 (16.53%)
Bot Behavior	Randomization		2,427 (40.29%)	6,141 (21.14%)	8,568 (24.43%)	1,623 (4.63%)
	Timing					6,945 (19.80%)
Total Cloaking Implementations			6,138	30,765	36,903	-

Table 3.4: Cloaking Technique Types in the APWG Dataset, as Detected by Crawlphish.

cation. From our crawling process, we found that the code obfuscation rate on phishing websites increased from 20.79% in 2018 to 24.04% in 2019. For example, for the Pop-up cloaking technique, the earliest variant from June 2018 was not obfuscated. Gradually, phishers started to obfuscate their cloaking technique implementations: in October 2018, they added an encoding algorithm, while the AST structure remained highly similar to un-obfuscated implementations. Later, phishers started to symmetrically encrypt client-side cloaking techniques (e.g., by using AES) and included decryption keys only as request parameters. In such cases, the AST of the same cloaking technique would differ from an existing group, so we place them in a new group. However, with CrawlPhish, we still find similar web API calls, so we consider this group to be an evolution of a prior group (its *origin*). From this finding, we gain the intuition that cybercriminals are improving client-side cloaking techniques in phishing to make the latest implementations more difficult to analyze.

3.6.4 Trends in Cloaking Usage

Table 3.4 shows the prevalence of each client-side cloaking technique type that CrawlPhish detected. Note that the sum of each cloaking technique's occurrence may exceed

Cloaking Technique			Total		Share	
Category	Type		Count	Percentage	Count	Percentage
Fingerprinting	Cookie		6,633	24.28%	2,912	9.87%
	Referrer				2,665	9.03%
	User-Agent				1,056	3.58%
User Interaction	Pop-up	Alert	17,634	64.55%	7,641	25.89%
		Notification			1,233	4.18%
	Click Through				6,735	22.82%
	Mouse Detection				2,025	6.86%
Bot Behavior	Randomization		5,294	19.38%	262	0.89%
	Timing				4,987	16.90%
Total Cloaking Implementations			29,561	-	-	-

Table 3.5: Cloaking Technique Types in the Public Dataset (September to December 2019), as Detected by CrawlPhish.

100% because some phishing websites implement multiple cloaking techniques. In the table, the percentage under the "2018", "2019", and "Total" columns represents the share of each *category* of JavaScript cloaking technique implementation in the respective time period. The percentage under the Share column refers to the percentage of each *type* of cloaking technique in all the cloaked phishing websites we detected.

We categorize the cloaking types in phishing websites from both the APWG Dataset and the Public Dataset. As shown in Table 3.4, the User Interaction cloaking category has the most implementations among phishing websites in the APWG Dataset. In 2018, 2,416 phishing websites (40.11%) leveraged cloaking within the User Interaction category, while in 2019, the usage ratio of User Interaction cloaking grew to 61.23%. The usage ratio of cloaking techniques in the Fingerprinting category over two years is almost the same. Within the Bot Behavior category, the usage ratio dropped significantly, from 40.29% to 21.14%. We find that phishing websites rely more on cloaking techniques in the User Interaction category than the others. We believe that this is because it is more difficult for anti-phishing crawlers to impersonate human behaviors than to bypass other types of cloaking.

2018			2019		
Targeted Brand	**Count**	**Share**	**Targeted Brand**	**Count**	**Share**
LinkedIn	2,317	38.46%	Apple	6,298	21.69%
PayPal	1,104	18.33%	Bank of America	3,572	12.30%
Microsoft	646	10.72%	Facebook	2,230	7.68%
Bank of America	309	5.13%	PayPal	1,841	6.34%
Apple	153	2.54%	Microsoft	987	3.40%

Table 3.6: Top Brands Targeted by Cloaked Phishing Websites in the APWG Dataset.

Table 3.5 demonstrates the usage of each cloaking type CrawlPhish detected from the Public Dataset. Just as we observed from the 2019 portion of the APWG Dataset, the User Interaction category was also the most frequently implemented in the Public Dataset.

Brand distribution. Among the 6,024 cloaked phishing sites in 2018, LinkedIn and PayPal were the most frequently impersonated brands, as shown in Table 3.6. In 2019, the distribution changed: Apple and Bank of America phishing websites were the most prevalent. Overall, four of the top five brands in 2018 were also in the top five in 2019. Nevertheless, because of changes within the phishing landscape between the two years, our findings regarding the relative distribution of cloaking phishing websites may be skewed.

3.7 Evaluation: Impact of Cloaking techniques

We have, thus far, shown that phishing websites make extensive use of client-side cloaking techniques. To demonstrate that this cloaking represents a significant threat to users, we deployed two experiments to verify that these techniques can truly evade detection by anti-phishing systems, and that they generally do not discourage victim visits—the two key factors to increasing attackers' return-on-investment.

3.7.1 Effectiveness Against Anti-Phishing Entities

We evaluate how effective client-side cloaking techniques are against real-world anti-phishing systems. Using a testbed for empirically measuring anti-phishing blacklists [5],

we first deployed 150 carefully-controlled artificial PayPal-branded phishing websites using new and previously unseen domain names: 50 for each of the top three User Interaction cloaking types we found in the wild (Notification, Click Through with a fake CAPTCHA, and Mouse Detection). We then simultaneously reported the URLs to key anti-phishing entities across the ecosystem (Google Safe Browsing, PhishTank, Netcraft, APWG, PayPal, and US CERT [4]) to evaluate if the ecosystem can collectively detect our cloaked websites. Lastly, we monitored the detection status (i.e., blacklisting) of our websites in major web browsers (Google Chrome, Opera, and Microsoft Edge, each powered by different detection backends) over seven days.

At the conclusion of these experiments, we found that *none* of our phishing websites were blacklisted in any browser, with the exception of Click Through websites, 21 (42%) of which were blocked in Microsoft Edge a median of 3 hours after we reported them. The detection occurred because Microsoft SmartScreen classified the obfuscation in the JavaScript source code as malware, not because it was capable of bypassing the cloaking technique itself. The fact that so many of our websites remained unmitigated after a seven-day period shows that client-side evasion methods are indeed effective at evading detection by modern anti-phishing systems.

Manual inspection is used by some anti-phishing entities [94]. Recurring suspicious websites that cannot be detected by automated systems should go to manual inspection for further analysis. With specialists' inspection, any malicious websites therein should be labeled as phishing and be blacklisted to protect users. Our observations, however, imply that our test phishing websites may have simply been classified as benign by anti-phishing systems and never sent for manual review. We believe that this is a clear limitation of current anti-phishing mitigations. Therefore, it is important for the whole anti-phishing ecosystem to understand the nature and prevalence of client-side cloaking techniques used by sophisticated phishing websites, especially when we consider the growth of such websites [71].

	Mouse Detection Count (%)	**Click Through** Count (%)	**Notification Window** Count (%)
Can See	879 (100.00%)	859 (97.72%)	374 (42.55%)
Cannot See	0 (0.00%)	20 (2.28%)	505 (57.45%)

Table 3.7: Experimental Results on the Effect of Cloaking Techniques on Users' Ability to See Phishing Content.

3.7.2 Hampering Victim User Traffic

To verify that client-side cloaking techniques in the User Interaction category do not significantly prevent users from being exposed to phishing content on cloaked phishing websites, we conducted an IRB-approved user study through Amazon Mechanical Turk [95]. Using a free hosting provider, we generated three websites: one with each of the same three types of cloaking as considered in the previous section (Notification, Click Through with a fake CAPTCHA, and Mouse Detection). Rather than hiding phishing content behind the cloaking, however, we simply hid the text "Hello World". By default, a blank page would be shown. We then hired 1,000 workers in Amazon Mechanical Turk and requested them to report what they saw after visiting each of the three websites [96]. We choose these three cloaking techniques because they are unique to client-side (rather than server-side) cloaking implementations, and because the other techniques have been tested in a server-side context [5].

Table 3.7 shows the detailed experimental results. 121 of the 1,000 workers could not view our phishing websites due to a technical problem: their browsers automatically added "www" in front of the sub-domains in our URLs, which may occur in older versions of web browsers [97]. Thus, the responses of 879 workers were suitable for analysis.

For the Mouse Movement cloaking technique, 100% of the workers saw the "Hello World" text, and thus would have also seen phishing content had they visited a malicious

website. For the Click Through websites, 97.72% saw the text, which shows that this cloaking technique is also effective against users. However, only 42.55% of the users saw the text on websites with the Notification Window cloaking technique. Nearly all users who did not see the text (94.94%) opted to *deny* notifications; the rest had incompatible browsers.

Although two of the cloaking techniques did not significantly prevent users from viewing the content, we found that the Notification Window cloaking technique has a negative impact on phishing success rates against potential victims. However, had these users been successfully deceived by a phishing lure (e.g., one that conveys a sense of urgency) prior to visiting the page, we believe that they would have been more likely to allow notifications [98]. Moreover, given the fact that websites with this cloaking technique were not detectable by the anti-phishing ecosystem (as we showed in Section 3.7), we still believe that this technique remains viable overall. In fact, the website shown in Figure 3.4 was still online in January 2020 even though we first observed the phishing URL in May 2019.

Consequently, we conclude that client-side cloaking techniques in the User Interaction category enable phishing websites to maintain profitability through a much longer life span, generally without discouraging victim visits, which in turn allows phishers to harm more users.

3.7.3 Responsible Disclosure

Once we established that the cloaking techniques discovered by CrawlPhish were capable of evading anti-phishing systems while remaining effective against human victims, we disclosed our findings, and the corresponding JavaScript code for each technique tested, to the major anti-phishing blacklist operators: Google, Microsoft, and Opera. All companies acknowledged receipt of our disclosure. Google followed up by requesting more information on the semantics and prevalence of the cloaking techniques, and concurred with

our finding that such techniques could potentially bypass detection by current automated anti-phishing systems.

3.8 Countering Client-side Cloaking Techniques

As we have observed, phishers make extensive use of sophisticated evasion techniques in their phishing attacks. The unique feature of client-side cloaking techniques is to require visitors to interact with the website or browser, such as through a button click or mouse movement. Phishers adopt such strategies because they believe that their victims will exhibit these behaviors when visiting a website [82]. If the website is in the process of rendering and shows a blank page, most people tend to move their mouse subconsciously. Similarly, out of habit, users will click a button from a pop-up or notification window to make web page content appear. We expect that phishers' degree of sophistication will only continue to grow. Therefore, the ecosystem should ensure that existing detection and mitigation systems are capable of adapting to such evasion techniques.

To detect advanced phishing websites with client-side cloaking techniques, anti-phishing crawlers should match the behaviors that sophisticated phishing kits expect. Specifically, crawlers need to impersonate human behaviors such as mouse movement and button clicks. To examine a given website, anti-phishing systems can emulate such behaviors using automated browsers. In addition, as we observed in our analysis, the *Notification Window* technique seems to exploit the lack of support for web notifications by current automated browsers. Thus, it is important for anti-phishing systems to close this gap and ensure that the browsers being used for detection support the same features as those used by potential victims.

Also, CrawlPhish can be directly incorporated into existing anti-phishing crawlers. With the hidden web page content revealed by CrawlPhish alongside traditional attributes such as URLs, we believe that current anti-phishing systems could identify malicious web-

sites that would otherwise evade detection. Furthermore, by implementing CrawlPhish analysis, crawlers would be able to more accurately classify and fingerprint new variants of evasion techniques employed phishing websites, or even discover entirely new types of cloaking. Such analysis would be particularly helpful in countering phishing websites that cannot currently be classified with high confidence.

3.9 Related work

Studies on phishing and cloaking techniques: Oest et al. analyzed server-side cloaking techniques within a dataset of 2,313 phishing kits and proposed a taxonomy of five different types of cloaking [4]. These authors also showed that cloaking techniques, including basic JavaScript cloaking, can effectively bypass detection by anti-phishing blacklists [5]. Based on an end-to-end analysis of large-scale phishing attacks, Oest et al. discovered that phishing websites with sophisticated evasion techniques are prevalent in the wild but the anti-phishing ecosystem has not effectively mitigated them [71]. In this work, we have presented the first in-depth analysis of client-side cloaking techniques in the context of phishing based on a dataset of 112,005 live phishing websites.

Invernizzi et al. studied server-side web cloaking techniques against search engines, and proposed mechanisms to identify and bypass such cloaking [7]. CrawlPhish leverages these methods to overcome server-side cloaking during crawling. The authors rooted their study in black markets and built a classifier to detect cloaking techniques implemented on the server side that returned different content to distinct browsing clients. This work mainly focused on the mutation of browser profiles to bypass server-side cloaking techniques to discover divergent web content. The authors found that 11.7% of search results were cloaked. The authors considered cloaking techniques used for Search Engine Optimization (SEO), advertisements, and drive-by download attacks. However, they did not investigate client-side cloaking techniques implemented in JavaScript (i.e., that execute in

the browser). In contrast, we discovered diverse client-side cloaking techniques and analyzed them from the perspective of phishing attacks.

JavaScript analysis techniques: Although a number of static analysis [99, 25, 23] and dynamic analysis [24, 22] approaches have been proposed to analyze malicious JavaScript code, there has been no attempt to automatically extract JavaScript code semantics for identifying and classifying cloaking techniques. Arrow and Zozzle are static analysis methods to classify JavaScript malware based on previously discovered malicious scripts [99, 25]. Revolver tried to detect evasive JavaScript code through similarity checks against known malicious matters [23]. Rozzle is a multi-execution virtual machine to explore multiple execution paths in parallel for enhancing the efficiency of dynamic analysis so that it can be used in large-scale experiments [24]. J-Force enhanced dynamic analysis methods to find hidden malicious behaviors by force-executing JavaScript code, regardless of the conditions, to explore all possible execution paths in an automated way [22]. Hence, J-Force lends itself well to revealing content hidden behind JavaScript cloaking code.

Analysis of program semantics similar to ours has been performed within other contexts. To deal with virtualization-based obfuscation, Coogan et al. proposed a de-obfuscation approach that identifies behaviors of malicious programs based on the flow of values to system calls [100]. BEAGLE assigns semantics to malware by dynamically monitoring system and API calls that malware uses to compare versions of malicious code and quantify their differences—to observe the evolution of a series of malware [101]. Zhang et al. introduced a semantic-based static analysis approach to reveal malicious Android applications' behaviors regardless of minor implementation differences [102]. The authors leveraged an API dependency graph to determine the semantics of the program to classify malware and identify malware variants.

3.10 Conclusion

Through the first in-depth analysis of the client-side JavaScript code used by phishing websites, we have uncovered a wide gamut of sophisticated evasion techniques used by attackers. In addition to categorizing such evasion techniques based on their semantics, our approach enabled us to measure the prevalence of each technique in the wild. In doing so, we observed that client-side evasion is becoming increasingly common.

Client-side JavaScript enables website developers to implement complex interactions between their websites and visitors. Thus, evasion techniques implemented in this manner pose a particular threat to the ecosystem: websites that use them can effectively discriminate between automated crawler visits and potential human victims. Unfortunately, client-side evasion techniques are difficult to analyze due to the dynamic nature of JavaScript code. CrawlPhish addresses this difficulty in a scalable manner. In addition to being able to detect and categorize client-side evasion with high accuracy, our approach can also track the origin of different implementations.

Given the rise of sophisticated phishing websites in the wild, we believe that automated analysis systems such as CrawlPhish are essential to maintaining an understanding of phishers' evolving tactics. Methodology such as ours can be incorporated by the ecosystem to more expeditiously and more reliably detect sophisticated phishing, which, in turn, can help prevent users from falling victim to these attacks through the continuous enhancement of the appropriate mitigations.

Chapter 4

UNDERSTANDING THE SCALY UNDERBELLY OF REDIRECTION CHAINS IN PHISHING

CrawlPhish can help the ecosystem analyze and detect phishing websites with client-side cloaking techniques, which is a symptom of this attack. However, there are phishing websites which implement other evasion techniques such as redirection to circumvent anti-phishing systems including CrawlPhish. So I present PHISHPATH to analyze phishing websites with redirection and help detect such advanced attacks.

Even though previous studies focused on the detection and analysis of *phishing lures* (e.g., emails and social media posts) [50, 103, 104], identifying phishing content in *landing pages* by using content-based features [33, 34, 35, 37, 29], and developing anti-phishing blacklists to prevent phishing by blocking known malicious URLs [27, 105], there still remains a significant gap in understanding the phishing ecosystem: The gap between when a user clicks on a phishing lure and the point at which they arrive on the landing page. In the online malware context, Stringhini et al. [106] discovered complex strategies, centering around the use of website *redirection chains*, that were leveraged by attackers to evade detection systems. Phishing seems ripe for similar use, but the use of redirection chains in phishing and, critically, their impact on the detection and prevention of phishing attacks, has not yet been studied.

By collecting and measuring redirection behaviors in phishing links in the wild, I can not only gain an understanding of redirection use, but also use this knowledge to improve existing anti-phishing systems and propose new mitigations. Understanding the prevalence and effectiveness of redirection in phishing is thus an important problem for improving the anti-phishing ecosystem.

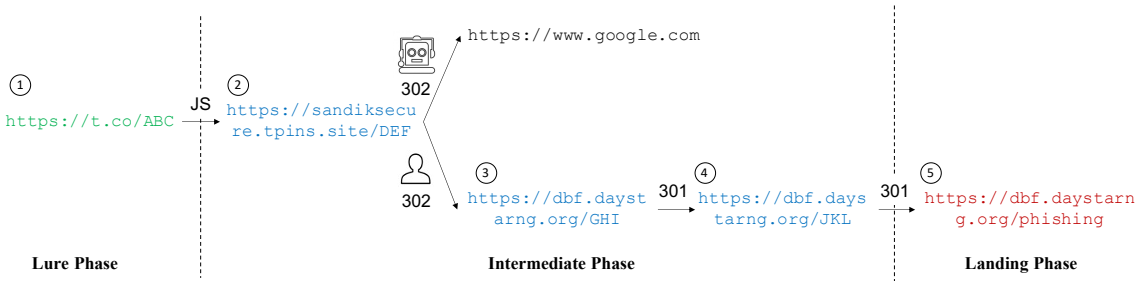


Figure 4.1: Example of a Phishing Redirection Chain: The lure phase (Green), intermediate phase (Blue), and landing phase (Red).

To this end, I propose PHISHPATH to perform the first large-scale analysis on redirection use in the modern phishing ecosystem by collecting and analyzing phishing emails and live websites. I ran my study for over a year from 2020 to 2021, collecting and analyzing 136,791 distinct phishing URLs from the Anti-Phishing Working Group (APWG), PhishTank, and victim-reported phishing emails from a frequent phishing-targeted organization. I discovered that *phishers are using redirection techniques* and that *current anti-phishing systems are not capable of effectively detecting phishing campaigns that use redirection links*.

I found extensive redirection use in phishing, with 34% of phishing attacks making use of redirection techniques. I identified a common pattern in the design of redirection chains, which I diagram in Figure 4.1: a *lure phase*, when a URL is embedded in phishing emails or social media posts that trick victims into clicking, after which the victim's HTTP request is redirected through the *intermediate phase*, where phishers identify anti-phishing bots and divert them to legitimate websites. Finally, after passing the server-side and client-side cloaking techniques in the intermediate phase, the visitors will see the phishing content in the *landing phase*.

While prior work studied phishing from the perspective of the lure phase and landing phase, I found that the intermediate phase is critical in the evasion of phishing detection

(Section 4.4.1). I found that attackers employ URL shortening services to help evade detection systems, including shorteners that *require user interaction* with a web page, rendering detection systems ineffective. Such phishing links were active (and not blacklisted) five months after initial detection. Moreover, I found that phishers exploit open redirect services in LinkedIn and Google to hide their malicious URLs.

Aside from the use of the intermediate phase for cloaking, I found that it also granted attackers additional flexibility in adapting to blacklist-based defenses. Because anti-phishing systems only blacklist specific URLs instead of domains, I observed phishers generating new URLs under the same domain to replace previously-blacklisted hops and keep the chain accessible for *as long as seven months*. In fact, I found that over 30% of phishing redirection chains *reused* domains in their intermediate layers.

I also measured the performance of anti-phishing blacklisting, online URL scanning, and feature-rich machine learning classification against redirection- and non-redirection-enhanced phishing attacks. Facing redirection phishing websites, these methodologies show degradation on the performance, compared with that against non-redirection ones. The difference in blacklisting [27] and online URL scanning [28] latency between the two groups is 6 hours and 42 minutes on average. Additionally, a state-of-the-art machine learning classifier [29] has a high false-negative rate (10.82%) and a low F1 score (0.87) when categorizing redirection phishing websites, compared with the 0.37% of false-negative rate and the 0.99 F1 score recorded in the previous work. These results demonstrate that redirection techniques effectively delay detection and classification.

Based on my analysis results, I propose two mitigation techniques. First, blacklist-based anti-phishing systems can block commonly reused redirection domains instead of only specific URLs, with few resulting false positives. This defense would end the currently trivial bypass of blacklists by attackers and reduce phishing attack profitability due to increasing malicious domain "burn rate." Next, I design a machine learning model based

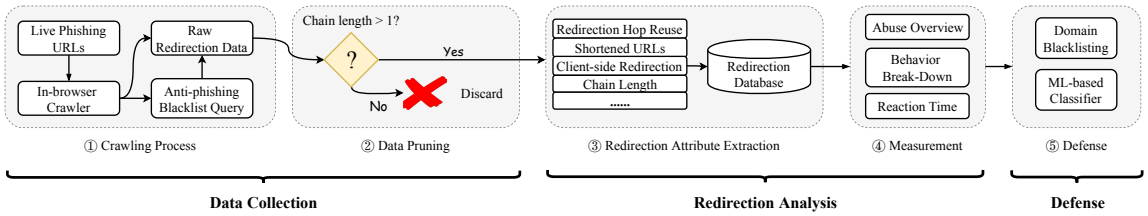


Figure 4.2: PHISHPATH Analysis Flow.

on attributes of redirection chains used in phishing websites to classify such sites without relying on content-based features in often-stealthy landing pages. The PHISHPATH classifier can identify phishing pages with only redirection information with high accuracy (90.76%) and a high F1 score (0.92). Finally, I discover that many entities are involved in redirection use in phishing. Based on this finding, I propose several ecosystem mitigations, including the tracking of redirection data, using redirection data to classify advanced phishing as well as locate lures, and restricting redirection services without identity information.

4.1 PHISHPATH Overview

We design PHISHPATH as a framework to understand and characterize the landscape of redirection techniques used by phishing campaigns in the wild. As illustrated in Figure 4.2, PHISHPATH contains two steps. First, PHISHPATH continuously crawls phishing URLs, collects redirection data, and filters non-redirection links (Section 4.2). Based on the collected data, PHISHPATH then conducts data analysis, extracting redirection attributes and investigating the characteristics in the redirection in phishing attacks (Section 4.3 and Section 4.4). Next, we demonstrate the impact of using redirection techniques on the anti-phishing ecosystem (Section 4.5). Finally, based on the analysis results, we propose two types of defenses (Section 4.6 and Section 4.7) against phishing websites using redirection techniques.

4.2 Data Collection

The inputs to PHISHPATH are phishing lure and page URLs. We collected URLs from the following three datasets [1] :

1. Phishing Email Dataset. From May 2020 to August 2020, we successfully crawled 10,564 distinct URLs directly extracted from phishing emails reported to a large industry organization that is commonly targeted by phishing. To avoid ethical concerns, the organization programatically extracted these URLs for the purposes of this research; we did not have access to sensitive information such as email addresses or the email body. 79.45% (8,393) of the URLs from phishing emails contain redirection.

2. Commodity Dataset. We also crawled malicious URLs from two commodity datasets: APWG eCrime Exchange Database [107] and PhishTank Database [92] — curated clearinghouses of phishing URLs maintained by various organizations engaged in anti-phishing. Both of the databases have a low false positive rate [108], which enables us to understand the nature of redirection techniques in phishing attacks. These datasets help gain a wide understanding of redirection use in phishing attacks in the wild from the perspective of different targeted brands and possibly different redirection behaviors. We first crawled 45,475 unique phishing entries from June 2020 to February 2021 from the APWG Database, which we refer to as *Legacy Commodity Dataset*. We then collected another 80,752 distinct URLs from June 2021 to September 2021, named as *Current Commodity Dataset*. As we notice, 32.50% (14,781) of the Legacy Commodity Dataset and 28.07% (22,671) of the Current Commodity Dataset contain redirection, compared with 79.45% redirection rate in the Phishing Email Dataset. We believe that this difference is due to reporters submitting the destination URL to the commodity databases, while people who receive phishing emails of a targeted organization report the email containing the lure URL.

[1] We open source the Commodity Dataset and the Benign Dataset: `https://mega.nz/folder/nSIggRYQ#p9WFzujoP3eKHYuKHITXkQ`

3. Benign Dataset. From June 2021 to September 2021, we crawled 15,128 benign redirection chains out of 27,162 URLs extracted from legitimate emails to compare the difference of redirection behaviors between both categories. We believe that benign redirection chains can be compared with the malicious ones because both groups of the redirection chains start with the entry hop in the lure.

Redirection Collection. To understand the intermediate phase in the phishing ecosystem, we collected redirection data for phishing campaigns in the wild. Through manual analysis, we observed that many phishing redirections use *client-side* redirection (e.g., JavaScript-based or `meta` HTML element redirections). Therefore, we design our crawler to support client-side redirection by opening the suspicious URLs in a real browser that executes JavaScript. With a modified browser extension [109], the crawler can collect the IP address, URL, redirection type (both server- and client-side), HTTP request, and HTTP response for each hop. PHISHPATH automatically collects the redirection information by crawling each reported URL, shown in Figure 4.2.

The PHISHPATH crawler analyzes each reported lure or landing page, traverses the redirection chain, and collects the URL, IP address, redirection technique used, reported date, and the raw HTTP response. PHISHPATH also continuously queries a blacklist based anti-phishing system and one online scanning engine to check when each URL is blacklisted or detected, and hence measures the latency. After automatic crawling, we identified 1,127 missing chains, due to redirection services that require manual clicks/intervention, such as *linktr.ee*. We manually investigate and fix all the missing chains.

Phishers use both server- and client-side redirection techniques in the wild. To capture the URLs generated by both types, we design multiple schemes to capture the redirections and label them by the associated technique. For server-side redirection, we inspect the HTTP response status where a `30*` code implies a redirection. For client-side redirection, we analyze the `meta` HTML element and related JavaScript code.

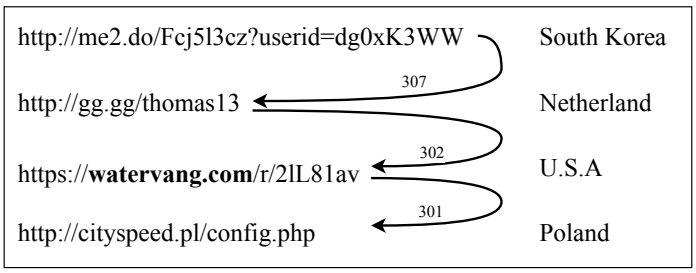


Figure 4.3: Example of Redirection Chain with Re-used Hops.

4.3 Redirections in Phishing Campaigns

To understand how redirections are used by phishing campaigns, we address the following research questions:

1. How often do phishing campaigns use redirection? (Section 4.3.1)
2. How well do anti-phishing systems detect redirection chains? (Section 4.3.2)
3. Do phishing campaigns use redirection in the same way that benign websites users do? (Section 4.3.3)
4. What is the relationship between redirection and cloaking? (Section 4.3.4)

4.3.1 The Prevalence of Redirection

We analyzed all 10,564 unique phishing URLs in the Phishing Email Dataset and found that 8,393 (79.45%) URLs pointed to malicious redirection chains. On average, each redirection chain has 3.73 hops. We found that phishing websites extensively reuse hops in their redirection chains. There were 1,692 unique domains in the intermediate and landing layers: only 20.1% of the 8,393 total redirection chains.

We extracted domain names from these phishing URLs and created **??** in Appendix. The bigger a node is, the more often its corresponding domain name is used by redirection chains. The largest two nodes (marked by black boxes) are each shared by over 500 chains.

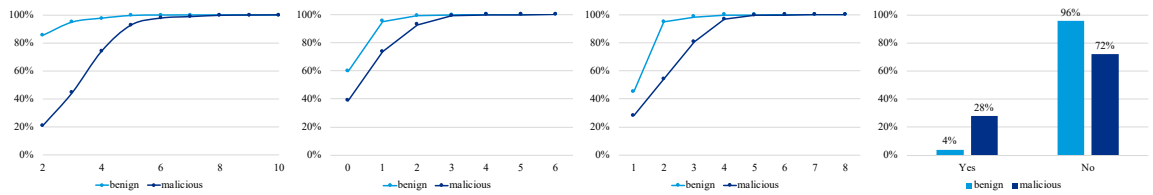
Upon closer examination, we found that these two shared domains are dedicated for redirecting traffic to phishing websites. Malicious redirection chains also use legitimate URL shortening services. Figure 4.3 shows a 4-hop redirection chain that uses one of the two domains, *watervang.com*, as well as legitimate shortening services, *me2.do* and *gg.gg*. All four hops are in different geo-locations.

Summary. Phishing campaigns frequently use redirection chains, and many chains share a large portion of domains.

4.3.2 Anti-phishing System Performance

Browser-based blacklisting is one of the main approaches to protect users from phishing websites. To study how malicious redirection chains impact anti-phishing systems, we first reported all the 8,393 phishing lures in the Phishing Email Dataset to Google Safe Browsing as we received them. We then observed how soon these URLs would be blacklisted. Within three days, Google Safe Browsing only blacklisted 289 out of 1,692 phishing domains, where 235 (81.31%) were landing hops. This indicates nearly 80% of these phishing websites successfully delayed or evaded detection by the state-of-the-art anti-phishing system. From May 2021, we also reported all 22,671 redirection chain lures in the Current Commodity Dataset when crawling to Google Safe Browsing. Within three days, Google Safe Browsing could blacklist 8,123 (35.83%) malicious redirection chains, where 59.33% (4,819) were destinations. It is possibly because Google Safe Browsing updated the blacklisting algorithm over the past year since our disclosure (Section 4.8.4) to block more entry and intermediate hops.

Since Google Safe Browsing uses content-based features to blacklist websites, it mostly blacklisted landing hops and left the majority of intermediate hops untouched. This behavior makes it easier for phishers to reuse intermediate domains in their malicious redirection chains.



(a) Chain length. (b) # of shortened links. (c) # of geo-locations. (d) Client redirection.

Figure 4.4: Comparisons Between Malicious and Benign Redirection Implementation to Quantify Indicators.

Additionally, we examined the performance of one of the online scanning engines, *urlscan.io* [28], by submitting 22,671 phishing websites with redirection from the Current Commodity Dataset. Only 954 were marked as malicious by *urlscan.io*. This evaluation also indicates that phishing websites with redirection can evade current anti-phishing systems.

Summary. Redirection chains allow phishing websites to significantly delay or even evade detection by anti-phishing systems. And anti-phishing systems like to blacklist landing hops more than phishing lures and intermediate hops.

4.3.3 Benign and Malicious Redirection Chains

We analyzed 46,192 unique phishing redirection chains in the Commodity and Phishing Email Datasets along with 15,128 benign chains in the Benign Dataset. Figure 4.4 shows clear differences in the characteristics of malicious and benign redirection chains. As shown in Figure 4.4a, the majority of benign chains (94%) has more than or equal to three hops, while half of malicious ones (55%) have more than three hops. Also, Figure 4.4b shows that many malicious chains heavily rely on URL shortening services (15% malicious and 2% benign chains use more than one shortened URL). Moreover, Figure 4.4c shows that only 5% of benign chains involve more than two geographically distinct IPs, but

Top Redirection Usage	Commodity Dataset	Email Dataset	Benign Dataset
Redirection hop reuse	31.06%	16.36%	–
Shortened link service abuse	24.42%	15.55%	1.48%
Client-side redirection	23.78%	28.44%	6.84%
Long length redirection	30.73%	55.82%	6.56%
Distinct geo-locations	28.26%	46.30%	7.18%
Open redirect exploit	7.13%	0.62%	–

Table 4.1: Percentage of URLs in Malicious (APWG and Phishing Email) and Benign Data Sets Demonstrating Each Feature.

Listing 4.1: Example of mandatory redirection in a phishing kit.

```
<?php
# -> All Created By [REDACTED]
# -> https://www.facebook.com/[REDACTED]
# -> ICQ : [REDACTED]
# -> // : Instructions
// You must use redirect or the scam
// will not open
$redirectlink = "l34kc0de.today";
...
```

over 55% of malicious ones involve three or more IPs in different geo-locations. Finally, benign chains rarely use client-side redirection techniques (Figure 4.4d).

Further, Table 4.1 shows a breakdown of each dataset in terms of percentages of redirection chains that exhibit each of the following features: redirection domain reuse, use of multiple URL shortening services, use of client-side redirection techniques, long redirection chains (in the rest of this paper, we define any chains with more than three hops as *long redirection chains*), involving IPs of multiple geo-locations, and use of open redirect exploits. The clear distinction of features between malicious and benign redirection chains convinces us that automatically differentiating them is possible, which encouraged us to develop a machine-learning based classifier as a defense (see Section 4.6).

Summary. Malicious and benign redirection chains exhibit significantly different features.

4.3.4 Redirection with Cloaking Techniques

Phishing websites heavily rely on cloaking techniques to evade the detection by anti-phishing systems [108, 4, 44]. We found that phishers usually implement cloaking techniques with redirection to display different content to different visitors.

To understand how redirections and cloaking techniques are used in tandem, we manually analyzed 512 phishing kits retrieved by *phishunt.io* [110] from May 2020 to July 2021. Four phishing kits mandate the use of at least one redirection hop (one such phishing kit is shown in Listing 4.1). In 198 of the examined phishing kits, redirections occur once the server side believes the visitor is a victim (instead of an automated detector), e.g., via `header("LOCATION: ../index.php")`. Hence, the use of redirections may indicate the existence of cloaking techniques in phishing attacks. Six phishing kits only redirect to real phishing pages after the visitor clicks a button on the phishing website. In this way, they reduce the risk of exposing phishing content to anti-phishing systems [104].

Additionally, we read posts in underground forums and discovered that these forums promote the use of malicious redirection, such as through the use of free VPS or rproxy servers from online service providers; criminals also exchange information on domains with open redirect vulnerabilities.

Summary. Phishing kits implement malicious redirection. Redirections usually occur with cloaking techniques.

4.4 Phishing Redirection Behavior

In this section, we analyze redirection behaviors in phishing to explain how phishers exploit limitations and vulnerabilities in the ecosystem through redirection.

Commodity Dataset			Phishing Email Dataset			Benign Dataset		
Service Provider	Count	Ratio (%)	Service Provider	Count	Ratio (%)	Service Provider	Count	Ratio (%)
s.id	1,681	32.53	me2.do	1,451	31.79	exct.net	881	25.33
linktr.ee	1,127	21.81	snip.ly	1,002	21.95	rebrand.ly	724	20.82
bit.ly	983	19.02	rebrand.ly	943	20.66	bit.ly	523	15.04
bitly.com	237	4.59	bit.ly	464	10.17	gg.gg	314	9.03
bit.do	204	3.95	s.id	241	5.28	s.id	286	8.22
snip.ly	172	3.33	zpr.io	82	1.8	goo.gl	246	7.07
rebrand.ly	170	3.29	bb3x.ru	70	1.53	snip.ly	152	4.37
tinyurl.com	170	3.29	gdy.club	68	1.49	ow.ly	123	3.54
rb.gy	124	2.4	gg.gg	58	1.27	attn.tv	121	3.48
me2.do	90	1.74	vk.cc	33	0.72	tinyurl.com	108	3.11

Table 4.2: Top URL Shortening Providers Abused by Phishing Attacks in Our Datasets.

4.4.1 Redirection Domain Reuse

We find that more than 30% of phishing redirection chains in the Commodity Dataset reuse redirection domains used in another chain, as shown in Table 4.1. Phishers share redirection services, which may be used to reduce the cost of phishing attack deployment [52] or to evade detection through previously deployed servers with cloaking techniques [108]. Thus, attackers can leverage existing infrastructure to avoid implementing redirection and evasion themselves.

By analyzing our results, in Figure 4.5, we summarize the top 10 shared domains in the redirection data that PHISHPATH crawled from the Commodity Dataset. Those common hops are responsible for 48.31% of reused domains in phishing campaigns in our dataset. This result shows that a small number of redirection technique implementations are likely responsible for a substantial portion of sophisticated phishing campaigns in the wild. Additionally, based on our analysis, phishers prefer to use REGRU-RU (31.06%) and No-IP (28.03%) to create malicious redirection services within the top 10 domains. Among the top 10 reused domains, *selcdn.ru* is reused the most among all domains: it is very versatile because it can be the domain of an entry URL, an intermediate hop, or a landing one.

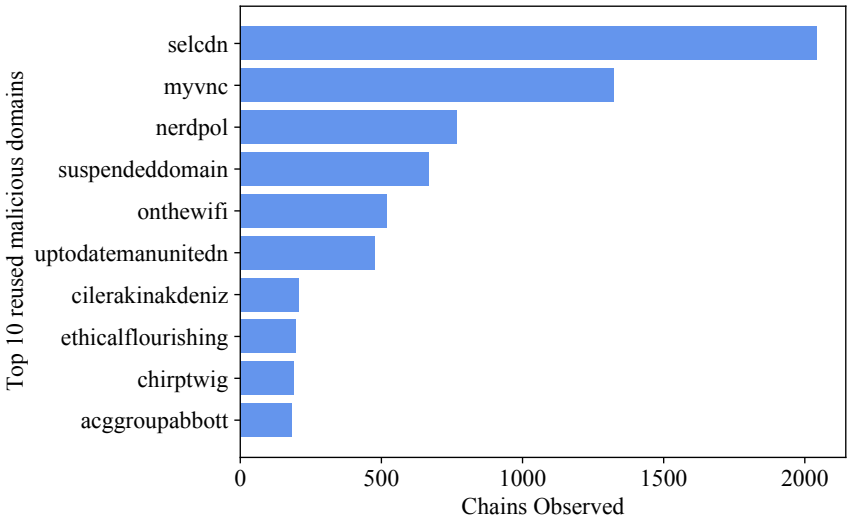


Figure 4.5: Most Used Redirection Domains in the Commodity Dataset. Long Domain Names Are Shortened by First Removing Their Tlds and Then Leaving the First 18 Characters.

However, according to our findings, anti-phishing systems such as Google Safe Browsing and *urlscan.io* did not blacklist or detect them until September 2021. Another popular domain is *myvnc.com*. Redirection chains using this domain contained seven redirection hops; while the entry URLs of these redirection chains have different domains, the intermediate and landing hops reuse this domain with different sub-domains and paths. The earliest observation of redirection chains using this domain is in May 2020, and the latest one is in November 2020. For the seven-month duration, despite URL blacklists such as Google Safe Browsing that detect the landing pages, new URLs with different paths were deployed every day. Therefore, while anti-phishing systems blacklist some destination paths, others remain accessible to potential victims, and the lures will work. The domain is now offline and was never marked as malicious by Google Safe Browsing.

Summary. Phishers reuse redirection domains to exploit the limitation that anti-phishing systems block only specific URLs instead of domains.

4.4.2 Malicious URL Shortening

To evade detection by the anti-phishing ecosystem, phishers use URL shortening services in malicious redirection chains. The first two hops in Figure 4.3 shows this in action—these services hide malicious redirection links and are used to bypass URL-based anti-phishing approaches [32, 33, 34, 35] so that malicious lures can reach the victim successfully. Additionally, phishing links with shortened URLs can appear benign to potential victims [111]. However, some URL shortening service providers strengthened their anti-abuse mechanisms [112], so phishers nest shortened links (as in Figure 4.3) to evade detection by using these shortening service providers.

As shown in Table 4.1, in the Commodity Dataset 24.42% of redirection chains use more than one URL shortening service. In contrast, the Phishing Email Dataset has similar evidence of shortened links usage: 15.55% contain more than one shortened links. Benign redirection links do not often include more than one shortened link: only 1.48% of them use multiple shorteners.

We inspect the top 10 shortening service providers in both datasets, shown in Table 4.2. In the Commodity Dataset, the top 10 URL shortening service providers account for 95.93% of malicious shortened link service use, compared with 96.67% of the usage in the Phishing Email Dataset. Among the used service providers, phishers prefer popular providers, such as *bit.ly*, *snip.ly*, and *rebrand.ly* in both datasets. We believe the following reasons are why phishers use URL shortening services:

Novel features used by phishers. *linktr.ee* usage is 21.81% in the Commodity Dataset, and its feature is displaying a list of links that the visitor is required to click. This feature is designed to help *linktr.ee* users guide their customers to specific websites, but it is used by phishers to evade detection as shown in Figure 4.6. To avoid detection, phishers misspell the *linktr.ee* account as *PayPa**l***. According to Zhang et al. [108], current anti-phishing sys-

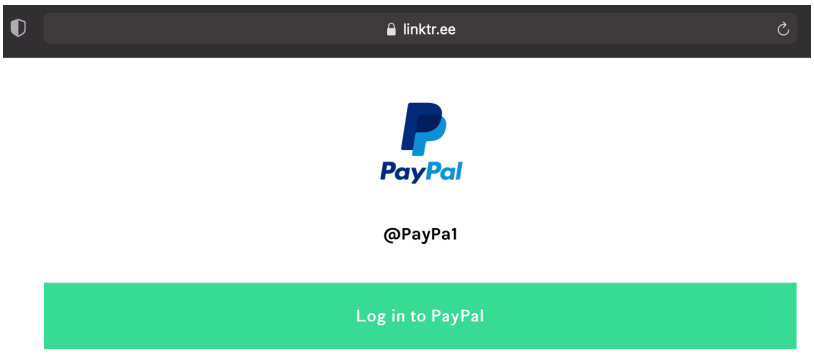


Figure 4.6: A Novel Feature in *linktr.ee* (`https://linktr.ee/PayPa1`) That Requires User Interaction.

tems cannot detect phishing websites preceded by a page with user interaction, so services such as *linktr.ee* are particularly dangerous due to their evasiveness. In fact, our collection system would have been unable to uncover these instances without our additional manual analysis. From the 1,127 phishing links redirected by *linktr.ee* in the Commodity Dataset, 20.88% of them were still live even five months after initial detection.

Redirection providers lack reporting channels. We find that some URL shortening providers do not offer any methods to report malicious redirections. Without reporting channels, it is difficult for providers to discover and remove malicious shortened links. Table 4.3 summarizes the malicious reporting type of top URL shortening services based on the observation in Table 4.2. For example, *bit.ly* and *rebrand.ly* only allow users to report suspicious shortened links [113, 114], while *snip.ly* does not provide any reporting options.

Summary. Phishers leverage required user interaction in URL shortening services to evade detection, and users do not have reporting channels for shortening services.

4.4.3 Redirection Chain Length

Our results demonstrate that 30.73% of the phishing websites with redirection in the Commodity Dataset use long redirection chains (defined in Section 4.3.3 as more than three

hops), while in the Phishing Email Dataset over 55% of redirection chains are long. Because phishing websites in the Phishing Email Dataset target only one brand, long redirection chains may be effective at evading detection, while phishing websites in the Commodity Dataset include a variety of brands, a less portion of which implement long redirection chains.

According to the CDF in Figure 4.4a, over 80% of malicious redirection chains have more than two hops. This may be because phishing redirection chains with only one basic cloaking technique, such as `.htaccess`, can be quickly detected within two hours [44]. Therefore, the improvement of detection systems forces phishers to improve their evasion techniques. We believe that phishers use multiple redirection hops in their phishing campaigns to either directly evade detection or introduce cloaking techniques from shared domains. We also find that 36.81% of the long malicious redirection chains contain reused domains. It is likely that phishers prefer to reuse existing cloaking implementations on dedicated servers rather than develop their own infrastructure. With multiple evasions implemented in one phishing attack, it is difficult for content-based anti-phishing systems to bypass all checks at all the hops and reach the final malicious phishing content.

Summary. The use of long redirection chains suggests that phishers need to implement advanced evasion techniques.

4.4.4 Open Redirect Vulnerability Exploitation

We find that phishers actively exploit open redirect vulnerabilities in the Commodity Dataset (7.13%), while they rarely exploit such vulnerabilities in the Phishing Email Dataset (0.62%). We determine that a domain is an open redirect vulnerability exploitation if it embeds another URL as a parameter. The difference between exploit rates of open redirect vulnerabilities in the two datasets results from the difference of phishing target.

Among those exploits, we find legitimate domains from services such as Google and

Top Service	Action			Blacklisted by GSB
	Report through email	Report online	No action	
linktr.ee		✓		✗
sniply.io			✓	✗
me2.do			✓	✗
rebrand.ly	✓			✗
s.id			✓	✗
bit.ly		✓		✗
tinyurl.com			✓	✗
cutt.ly				✗
rb.gy		✓		✗
zpr.io			✓	✗

Table 4.3: Abuse Reporting Types of Top URL Shortening Services.

LinkedIn: 467 (17.48%) of open redirect exploits were from LinkedIn and 255 (9.55%) of them came from Google, including Google Drive and Firebase. Phishers exploit LinkedIn open redirect vulnerabilities [2] to redirect traffic through a shared domain *onthewifi.com*, which is also observed in Figure 4.5. In the Phishing Email Dataset, we find that phishers use Google's *Firebase* service 1,081 times to create a dynamic redirection.

Summary. In malicious redirection chains, phishers exploit open redirect vulnerabilities in prominent websites.

4.4.5 Miscellaneous

Other redirection behaviors in Table 4.1 also appear in malicious redirection chains. Multiple geo-locations may result from phishers using different redirection infrastructure. According to our analysis of malicious and benign redirection chains in Section 4.3.3, we define a redirection chain as having multiple geo-locations if it contains more than two different geo-locations. We also define that two hops have different geo-locations if their IPs are in different States/Provinces. For example, phishers reuse shared domains or use

[2] An example of such URL is *https://www.linkedin.com/<READACTED>?url=http%3A%2F\%2Fwkei281321%2Emerlontean%2Enet&urlhash=DsM6&trk=public_profile-settings_topcard_website?idtrack=ueDW1xfL*.

URL shortening services to redirect the traffic to their own landing websites, both of which may contain hops from different geo-locations. We observed that 71.26% of redirection chains with reused domains have more than two distinct geo-locations.

Phishers also implement client-side redirection in their malicious redirection chains. We believe that use of client-side redirection is due to increases in client-side cloaking techniques [108]. After client-side cloaking techniques, phishers can either show or hide the web page content through JavaScript execution, or redirect the visitor to different websites by changing `location.href`. Besides, client-side redirection can help phishing websites evade anti-phishing crawlers that do not support JavaScript execution.

Summary. Phishers use multiple geo-locations and client-side redirection in malicious redirection chains.

4.5 Impact of Malicious Redirection

Thus far, we have demonstrated that phishing attacks extensively use redirection techniques and their redirection chains have distinctive features from benign ones. In this section, we show that the use of redirection in phishing can be a significant threat to the anti-phishing ecosystem. To this end, we selected one blacklist-based anti-phishing system (Google Safe Browsing) and one online scanning engine (*urlscan.io*) to evaluate the impact of malicious redirection chains [115, 116].

Specifically, we evaluate the ecosystem's *blacklist/detection speed* on the examined phishing websites. We define the blacklist/detection speed of an anti-phishing system as the duration between when one phishing URL is reported and when the anti-phishing system blacklists/detects the URL. We submitted each URL in the Current Commodity Dataset to Google Safe Browsing (GSB) and *urlscan.io* at the same time that the PHISHPATH crawler visited them. Then we periodically (every 15 minutes) queried the inspection result from both anti-phishing systems. Because we discovered that phishers combine redirection and

Redirection Technique	Mean Speed	Median Speed	Avg. Delta w/ Non-redi	# of Chains Over Avg. Delta	# of Chains Not BL/Detected	Total # of Chains
Long chain	10:45	09:02	07:16	1,527	3,831	5,742
Hop reuse	10:26	09:13	06:57	1,351	3,323	5,207
Client redirection	09:46	07:32	06:17	1,978	4,798	7,435
Shortened link	10:29	08:10	07:00	1,429	4,072	5,924
Geo-locations	10:34	09:39	07:05	1,240	2,820	4,546
Open redirect	09:33	03:57	06:04	203	1,549	1,943
All techniques	10:11	08:22	06:42	5,969	14,633	22,671
Non-redirection	03:29	02:52	00:00	–	–	–

Table 4.4: Blacklist/Detection Speed by the Ecosystem in hh:mm Break-down by Redirection Techniques in the Current Commodity Dataset.

cloaking to evade detection, we cannot separate these two techniques to measure the impact of only redirection. To compare reaction time between non-redirected URLs and redirected chains, we also measure reaction time of non-redirected phishing URLs.

Table 4.4 shows the blacklist/detection speed. In the Current Commodity Dataset, the average blacklist/detection speed for redirected phishing URLs is 10:11 (hh:mm) and the median speed is 08:22. Among them, 14,633 are not blacklisted or not detected by the anti-phishing systems, probably because the phishing URLs successfully evade detection. We find that non-redirection phishing URLs are blacklisted/detected with an average reaction time of 03:29, which is much faster than redirection-enhanced ones. This result highlights the danger that malicious phishing redirection chains can significantly reduce an anti-phishing system's blacklist/detection speed.

We further analyze redirection chains and the blacklist/detection speed in terms of redirection behaviors, and we discover that using every redirection technique reduces the blacklist/detection speed. A long redirection chain reduces the blacklist/detection speed of anti-phishing systems most, probably because cloaking techniques in the long chain makes analysis difficult. Redirection chains using URL shortening services also have a high impact on the blacklist/detection speed from the anti-phishing ecosystem, which is due to the

Model	Precision (%)	Recall (%)	FPR (%)	FNR (%)	F1	ACC (%)	ROC-AUC	Proc Time (s)
PHISHPATH (regular)	92.28	92.26	11.48	7.74	0.92	90.76	0.95	35,985
PHISHPATH (w/ sub-chain)	86.78	92.28	15.76	7.72	0.89	88.49	0.94	**6,797**
CANTINA+ (regular)	85.84	89.18	19.57	10.82	0.87	85.42	0.95	23,133
CANTINA+ w/ PHISHPATH	**98.95**	**98.64**	**1.53**	**1.36**	**0.99**	**98.57**	**0.99**	59,118

Table 4.5: Evaluation Metrics of Evaluation of Our Classification on Websites with Redirection.

non-disclosure and detection gaps between anti-phishing systems and URL shortening service providers. This result demonstrates that the high usage of URL shortening services is because of its effectiveness of extending the "golden hour" of phishing [104].

Summary. The decreased blacklist/detection speed from the anti-phishing ecosystem due to malicious redirection chains indicates that phishers use redirection in the intermediate phase of the phishing process to delay detection and sometimes evade anti-phishing systems.

4.6 Machine Learning Based Classification

Our analysis of differences in the redirection features between benign and malicious phishing redirection chains demonstrates that those features can be leveraged in a machine learning classifier to detect attacks. Classifying a lure as malicious using only the redirection chain would augment content-based anti-phishing systems that have difficulty identifying maliciousness due to cloaking techniques [108, 7].

In this section, we describe our classifier design, show that it is effective at identifying malicious redirection chains, and demonstrate how it can detect partial chains and be used to augment existing classifiers.

	Attributes (in a redirection chain)		
	Name	*Type*	*Explanation*
Redirection Type	301	Numerical	Number of Moved Permanently code
	302	Numerical	Number of Found code
	303	Numerical	Number of See Other code
	307	Numerical	Number of Temporary Redirect code
	308	Numerical	Number of Permanent Redirect code
	JavaScript	Numerical	Number of JavaScript redirects
	meta	Numerical	Number of meta redirects
Derived Attribute	chain_length	Numerical	Length of the chain
	hop_reuse	Numerical	Number of reused domains
	shortened_links	Numerical	Number of shortened links
	distinct_geo	Numerical	Number of distinct IP/geo-locations
	open_redirect	Numerical	Number of open redirect vuln. exploited
Output	label	Categorial	Binary: benign or malicious

Table 4.6: 12 Features That PHISHPATH Uses for Prediction and the Output Label of the Prediction.

4.6.1 Feature Selection

We first extract features from redirection behaviors as *redirection features*, shown in Table 4.6. The first group of numerical attributes—*Redirection Type*—refers to the number and type of client- or server-side redirect methods in one redirection chain. Server redirection HTTP status codes in benign redirection chains vary from those in malicious ones, so we treat different status codes as different features, unlike the work from Stringhini et al. [106].

Derived Attribute represents advanced attributes calculated from the raw redirection data, and include length, number of reused hops in the chain, amount of shortened link services, and distinct number of geo-locations of a redirection chain.

Our "output" layer represents intelligence we collect from the anti-phishing ecosystem about the analyst-determined maliciousness of the (landing pages at the end of the) redirection chains.

Evaluating features. To understand the impact of redirection features to PHISHPATH's

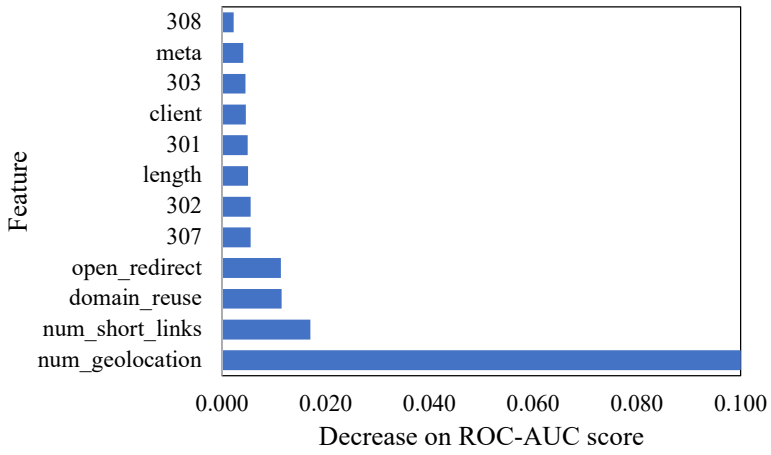


Figure 4.7: Feature Importance Indicated by Decrease on ROC-AUC. The Decrease Is Calculated by Subtracting Permutated ROC-AUC from Baseline ROC-AUC (0.954). Predictive Features Are Mainly from Advanced Feature Groups.

classifier, we evaluate the importance of each feature used in the model. The evaluation metrics used in our case is decrease on ROC-AUC score, because it is a good indicator for a model's comprehensive ability. The impact of each redirection feature on the decrease of ROC-AUC score is shown in Figure 4.7. *num_geolocation* and *shorten_links* features appear to have the most impact on the decrease of ROC-AUC score, meaning that these features are strong indicators to distinguish malicious and benign redirection chains. This matches our intuition that a chain with various geolocations or multiple shortened links in the redirect chain indicates its maliciousness, as discussed in Section 4.3.3. Other features, such as *open_redirect* and *hop_reuse*, also provide strong impact on the classification model.

4.6.2 Model Selection

We define *malicious* as 1 (positive) and *benign* as 0 (negative). To find the optimal machine learning algorithm for PHISHPATH's classifier, we randomly selected benign and malicious websites with redirection, 1,000 each, to train and test five machine learning

algorithms (train:test ratio is 8:2). The five algorithms are Logistic Regression (LR), K-Nearest Neighbor (KNN), Classification and Regression Trees (CART), Random Forest (RF), and Support Vector Machines (SVM). All five algorithms have an at least 88% accuracy rate (four of them even have an accuracy rate above 90%), which implies that the redirection features can explicitly help the machine learning models distinguish benign and phishing websites. Among all five algorithms, RF has the best accuracy rate. Hence, we selected RF as the classification algorithm. We then randomly selected another 2,000 websites with redirection, equally distributed on malicious and benign sides, and used grid search and five-fold cross-validation to tune the RF model parameters.

Websites with redirection in both the Benign and Current Commodity Dataset are included in the machine learning procedure. We choose these two datasets because they both fall into the same time frame (from May to September 2021). Following a 8:2 training:testing rule, the dataset used for training and testing contains, respectively, 30,239 and 7,560 redirect chains. As shown in Table 5.5, with the tuned parameters and the RF model, our classifier achieves an overall accuracy of 90.76% with an FP rate of 11.48% and an F-1 score of 0.92, *using only redirection data*, demonstrating the promise of our approach in detecting advanced phishing websites [3] .

Validation. To validate the effectiveness of the PHISHPATH classifier, we selected 500 previously unseen phishing redirection chains newly crawled from APWG and PhishTank and 500 benign ones from Common Crawl [117]. The accuracy rate reached 89.59% along with an F1 score to 0.89. The validation evaluation result demonstrates the effectiveness of the PHISHPATH classifier.

[3]We open source the training and test datasets along with the model: `https://mega.nz/folder/napBRCiR#iNTn9t5szqMJLK2aDgzWFA`

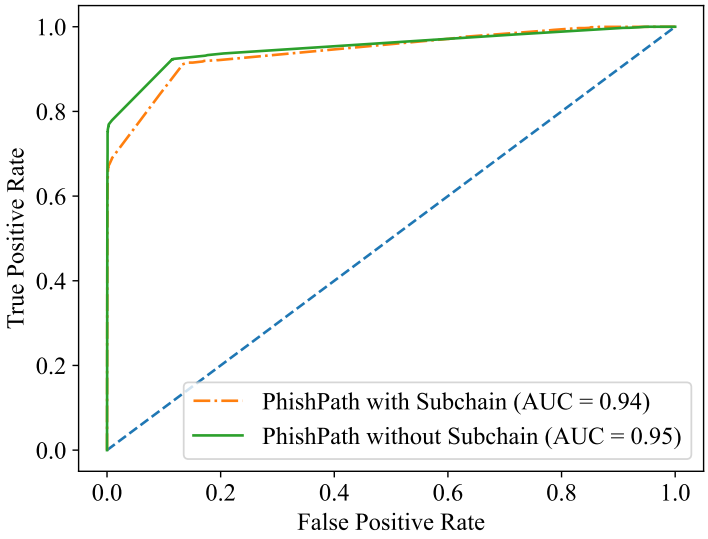


Figure 4.8: ROC Curves of PHISHPATH with and Without Sub-chain Mechanism.

4.6.3 Early Classification of Sub-chains

In the basic design, the classification system must visit *all hops* of a suspicious redirection chain to collect redirection information and then classifies. This design can have high accuracy because it has full information of the chain. However, the processing time for crawling one hop is 4.72 seconds on average in our experiments, which may negatively impact timely classification in practice for the real-time requirements of current anti-phishing systems.

Therefore, we also design a detection mechanism for early detection of suspicious redirection chains. We input *sub-chains* of a full redirection chain into the classifier to identify maliciousness *before crawling* the whole chain. The idea of the *sub-chains* mechanism is that the classifier is trained on redirection chains with a different number of hops, therefore, it is also reasonable to assume that the classifier is generalizable to capture features and can make an early classification when fed a sub-chain.

For example, consider a malicious redirection chain that contains five hops. When the

system inspects this suspicious chain, it crawls the first two hops of it as a two-length sub-chain, and then classifies whether it is phishing. If so, it stops crawling and saves execution time for future chains. Otherwise, the framework will crawl the next hop and classify the three-length sub-chain, until the full chain is crawled. Therefore, this technique can improve scalable for anti-phishing systems that require fast processing time.

Evaluating sub-chain detection. To evaluate the quick classification with *sub-chain*, we conduct an experiment to measure how much processing time on inspecting suspicious redirect chains PHISHPATH can save with *sub-chain* mechanism. In our test dataset, we still use the same test dataset in Section 4.6.2, containing 7,560 redirection chains with 16,371 hops. After eliminating chains with length of two (otherwise, the sub-chains will not actually be chains, but single hops), we have 1,982 redirection chains with 7,544 hops for both malicious and benign URLs. 1,400 (70.63%) of the total redirection chains are classified before PHISHPATH needs to crawl a complete redirect chain. With *sub-chain* mechanism, PHISHPATH only needs to crawl 1,425 hops. According to our experiments, average crawling time for one hop is 4.77 seconds. Thus, while the full-chain configuration of PHISHPATH needs 35,985 seconds (4.77 $\times$ 7,544) to finish crawling and classifying redirect chains, the *sub-chain* mechanism brings the runtime down to 6,797 seconds (4.77 $\times$ 1,425), saving 81.11% of the original processing time. [4]

To illustrate the difference, we conduct a comparison experiment between PHISHPATH with *sub-chain* and without *sub-chain*. The ROC curve of the modified classifier is shown in Figure 4.8. While the sub-chain classifier is effective, it is markedly less effective than the full-chain, with a 15.76% false positive rate. This false positive rate makes PHISHPATH's subchain mechanism unusable on its own, but its high precision (92.28%) and recall (92.26%) would make it a good pre-filter to inform defense systems in the use of

[4]We open source the training and test datasets along with the model: *https://mega.nz/folder/CSpmkTLT#NyxXX6iparTsEKcuYbrraw*

higher-overhead analyses.

4.6.4 Complementing Current Systems

Previously, we have shown that PHISHPATH is effective at detecting phishing attacks based solely on features of their redirection chains. However, in actual usage, more information than just the redirection chain can be used to make the classification decision. To explore this, we evaluate PHISHPATH's efficacy when running in tandem with an existing ML- and content-based phishing classifier, CANTINA+ [29].

We collected the features proposed in CANTINA+ of our malicious and benign redirection chains to demonstrate PHISHPATH's potential for this evaluation. In this configuration, CANTINA+'s content-based features are essentially disabled, as cloaking prevents the crawler from reaching the landing page [5] .

Following a 8:2 training:testing ratio, we train CANTINA+ and CANTINA+PHISHPATH (a union of the features from CANTINA+ and from PHISHPATH) with the same amount of websites with redirection as the evaluation in Section 4.6.2. As displayed in Table 5.5, CANTINA+PHISHPATH has a significantly higher detection ratio, achieveing a 98.57% accuracy rate and a 0.99 F1 score, compared to a 85.42% accuracy rate and 0.87 F1 score for CANTINA+ alone.

Failure of the retrieval of final web page content of phishing websites is likely the main reason why CANTINA+ has limited performance against advanced phishing websites. However, the CANTINA+PHISHPATH classifier allows it to reason about the path the victim takes toward the landing page, even if the landing page is unreachable. It, thus, can out-perform CANTINA+ over phishing links with redirection.

[5] We open source the CANTINA+ features: *https://mega.nz/folder/ObBG3ToT#bsXRr-JSCLDYXbPblrIayA*

4.7 Blacklisting Less, Blocking More

Our machine learning classifier predicts malicious redirection chains with high accuracy. While the classification result allows blacklisting-based anti-phishing systems to block *destination* phishing websites, it will cause the blocking of a large number of URLs, making it easy for phishers to revive phishing campaigns by swapping only the last hop. For example, in the Phishing Email Dataset, blacklisting destination websites would block 8,393 landing URLs.

Through analyzing data in PHISHPATH, we found that the only domains that are common to both malicious and benign redirection chains are URL shortening services and web hosting providers. Other commonly reused domains by malicious chains can be safely blacklisted. Therefore, we propose an improvement on blacklisting: *Blocking malicious intermediate domains that are dedicated to malicious phishing redirection chains*. This approach will blacklist *more* phishing websites with *less* blocklist entries while *increasing the cost* for phishers to maintain malicious redirection chains. It will effectively disable the intermediate layer.

Accurately identifying these malicious intermediate domains is possible since PHISHPATH has a global view on intermediate hops in both benign and malicious redirection chains. We first compared 699 domains in the Benign Dataset with 1,692 domains in the Phishing Email Dataset, which collectively account for 8,393 malicious redirection chains. We found only 42 common domains across two datasets, 24 of which are either URL shortening or web hosting services. The other 18 domains either had an open redirect vulnerability (such as LinkedIn) or are legitimate destination domains. These 42 domains should be excluded from domain blacklisting.

Next, we analyzed the 8,393 malicious redirection chains to find the minimum number of intermediate domains needed to be blacklisted to disable all malicious redirection chains,

excluding URL shortening and web hosting domains. We found that 675 of 1,692 domains in the Phishing Email Dataset would need to be blacklisted. Additionally, we manually inspected these 675 domains and found four false positives. Among them, three were compromised domains exploited by attackers to host phishing servers, and one was a false positive in the Phishing Email Dataset. Thus, our improvement on blacklisting can reduce the blacklist size by 92.4%, from 8,939 to 675 records, based on our dataset. Doing so blocks 5,377 more phishing websites, or 178% on top of the real-world blacklist of 3,016 URLs that we identified from Google Safe Browsing.

Finally, we analyzed the level of blacklisted domains in redirection chains. Six are phishing lures. 178 are in the first level of the intermediate layer, and the rest are in the second level. This shows that the proposed blacklisting mechanism can block malicious redirection chains within three hops.

4.8 Discussion

In this section, we consider our findings and reason about what actions might be taken to mitigate the threat posed by redirection chains in phishing.

4.8.1 How Did We Get Here?

The use of redirection chains to hide phishing campaigns seems in line with expectations in the field. Given this, why has the ecosystem (comprised of many companies that suffer from these attacks) not evolved mitigations?

Collaborative failures. The main reason, of course, is the required collaboration between the many entities in the ecosystem, which is true of phishing in general [44], but even worse when redirection chains are concerned, as this adds even more organizations into the mix. For example, a redirect chain could hop through public URL shorteners, exploit open redirect vulnerabilities in public websites, and use attacker-deployed redirection domains.

Detection and blocking of such a chain would involve entities ranging from anti-phishing systems (e.g., Google Safe Browsing), to URL shortening service provider (*bit.ly*), to web hosting provider, and even to domain registrars. All of the entities have to act, in tandem, to effectively block the phishing attack.

For a number of reasons, the necessary collaboration to facilitate such action does not exist. As a result, as we observe and measure throughout the paper, remediation of redirection-using phishing attacks happens in a piecemeal way, leading to a quick resumption of phishing activity as attackers simply route around the "damage."

Individual failures. Even on the level of individual entities, we identified redirection-induced limitations in anti-phishing systems. For example, Google Safe Browsing does not crawl data if a redirection chain is over five hops long [118], leading to evasion potential by phishing campaigns. Even when a redirection chain is short enough, Google Safe Browsing blacklists *the landing page* of the whole redirection chain, rather than lure or intermediate URLs in our datasets (this appears to be due to the risk of over-blacklisting [119]). As a result, phishers can easily "resurrect" an otherwise-burned lure by updating the destination of any of the intermediate redirects.

These gaps in individual services also reduce collaboration potential. For example, because Google Safe Browsing tends to blacklist only the landing page of a phishing campaign, legitimate URL shorteners used by phishers to redirect to other intermediate hops cannot effectively check the safety of the redirect [120].

Wide availability of phishing service resources. Phishers develop and share redirection hops for easy attack deployment and low cost [121, 52], abusing domain registrars, web hosting services, or even compromised servers to achieve this goal. Many of these services require only a credit card [122], and many provide their services for free. The availability of compromised and anonymized credit cards, used by cyber-criminals in their operations [123], makes these services ripe for misuse in phishing redirection chains. Though

many of these providers offer channels for users to report abuse, it takes an average of three days for them to respond to the report, giving phishers time to phish potential victims [124].

Vulnerable websites. Phishers exploit open redirect vulnerabilities in legitimate websites as part of their redirection chains to evade anti-phishing crawlers, automated filters, and even human scrutiny [125, 126]. These vulnerabilities are common on the web, with 46 assigned CVEs in 2020 alone [127] and an average time-to-fix of 34 days [128], meaning that multiple such vulnerabilities tend to be active (and exploitable by phishers) at any given time.

4.8.2 Where Do We Go?

Having reviewed the contributing factors behind redirection chain misuse in phishing, we provide several recommendations about potential mitigation techniques across the different entities involved.

Anti-phishing blacklists. Anti-phishing systems must move toward domain-level, rather than URL-level blacklists, as our results demonstrate the reuse of malicious redirection domains by phishers. This will force phishers to acquire additional domains, reducing the profitability of their attacks. As discussed in Section 4.7, this defense is more effective than current blacklist techniques and can maintain a low false positive rate.

Anti-phishing crawlers. Server-side anti-phishing systems are impacted by the widespread use of cloaking techniques [108, 5], which often foils the retrieval of the landing page by crawlers. In this case, the sub-chain mechanism of PHISHPATH can allow these systems to reason about maliciousness even when the landing page fails to be reached, blacklisting malicious intermediate hops even in the face of successful evasion. Interestingly, Google's Suspicious Site Reporter [129] has started collecting redirection data from the client side, though it is not clear how this data is currently used.

Web browsers. Major modern web browsers use client-side classifiers to identify mali-

cious pages [130], because evasion techniques, by definition, cannot be used to mask the landing page from victims (otherwise, the victim will not see it). However, these classifiers are oblivious to redirection chain data, and evasive measures can be taken by the content of the landing page to hide its maliciousness [43]. Augmenting existing browsers with our PHISHPATH ML classifier, as described in Section 4.6, would significantly improve the protection of web users against phishing.

Redirection service providers. In Section 4.4.2, we show that phishers abuse URL shortening services due to the lack of usage monitoring and user report channels. Also, as shown in Table 4.3, blacklist-based anti-phishing systems tend to not blacklist URL shorteners. Phishers use the interplay between these two situations to build effective, evasive redirection chains. To avoid large-scale abuse, URL shortener providers must implement up-to-date anti-phishing mechanisms.

4.8.3 Limitations

Though we shed light on serious problems limiting the modern ecosystem's ability to identify and prevent phishing attacks, there are a number of limitations to our work. In this section, we discuss these limitations.

As discussed in Section 4.3.3, the threshold of redirection features are selected empirically, by comparing features between the Benign and Phishing Email Datasets. However, redirection behavior in malicious and benign redirection chains may change over time, leading to degradation in PHISHPATH's ML-based classifier. This challenge is straightforward to tackle: PHISHPATH can continuously collect redirection data from phishing and benign URLs, measure current redirection trends, and refine its classifier with new data.

Data evasion attacks. Criminals could leverage techniques from Adversarial Machine Learning, tuning redirection implementations in phishing attacks to mimic benign redirections and bypass PHISHPATH's classifier. Though such evasions might have success,

this represents a fundamentally different environment than the freedoms phishers currently enjoy regarding redirection chains. Instead, it would force trade-offs onto phishers. For example, to weaken our classifier, phishers would likely have to remove cross-chain hop sharing, which, as discussed in Section 4.4.1, currently provides cost reduction and simplicity.

Data source selection. We leverage two data sources, the Commodity Dataset and the Phishing Email Dataset, as discussed in Section 4.1. The former contains phishing data for attacks targeting various organizations, while the latter lets us extract lure URLs directly from phishing emails. Together, this allows us to collect full redirect chains and to measure the redirection abuse targeting specific organizations.

However, reporters may submit only the landing URL of a chain to APWG or PhishTank, which may dilute our reasoning about redirection abuse rate. There are additional anti-phishing resources such as PhishLab that can be used as data sources, which can help mitigate this dilution. However, this dilution can only cause us to *under*-estimate rather than over-estimate the prevalence of redirection chains in phishing.

Potential biases in data source. There is a possibility that reporters may submit URLs of intermediate hops to APWG, leading to inconsistent features and classifier confusion. This is something that we can measure by leveraging duplicate submissions to APWG: if we check all submitted URLs against all intermediate hops seen in all redirection chains crawled by PHISHPATH, we can identify direct submissions of intermediate hops. In fact, analyzing our data, we found no such occurrences. We believe that it is mainly because redirection happens quickly when transferring users from the lure to the landing layer, leaving no chance for the victim to capture intermediate URLs to report.

Limited phishing kit availability. Phishing kits are difficult to acquire unless the malicious server is taken down by anti-phishing infrastructures. Therefore, with limited amount of phishing kits retrieved by *phishunt.io* [110], our analysis of redirection setting in phish-

ing kits may not comprehensively cover all possibilities and implementations. However, the wide abuse of redirection techniques observed from both datasets help us demonstrate that redirection is widely implemented in phishing kits.

4.8.4 Responsible Disclosure

Once we established that the redirection behavior measured by PHISHPATH was capable of facilitating phishing campaign deployment and helping evade anti-phishing systems, and determined that PHISHPATH can classify by only using redirection features, we disclosed our measurement, machine learning based classifier, and ecosystem recommendations to the major entities in the anti-phishing ecosystem. These were Google, Linktr.ee, bit.ly, and Namecheap, because they are the representatives in blacklist-based anti-phishing systems, URL shortening services, and web hosting/domain registrars according to our analysis above. Google and Linktree acknowledged receipt of our disclosure. Linktree did not have a further response; Google followed up by requesting detailed redirection data for the reported phishing URLs, and we believe this may have contributed to the change in the behavior of Google Safe Browsing, as discussed in Section 4.3.2.

4.9 Related Work

To mitigate large-scale phishing, the research community has conducted a surge of research. However, the prevalence and nature of redirection abuse in phishing has not been thoroughly studied yet.

Oest et al. [104] discovered that sophisticated phishing attacks are causing substantial damage to users. Especially, redirection techniques with cloaking techniques are a key component of advanced phishing websites to hamper detection [108, 7]. Redirection, also, distributes benign-looking URLs to potential victims, routing them through complex chains [131, 132].

To investigate malicious use of redirections, Bhargrava et al. [133] compared legitimate and malicious redirections and found that spammers leverage client-side redirections more often than server-side ones. Also, Stringhini et al. [106] proposed complex strategies for the use of redirection chains in the context of drive-by-download attacks. Chellapilla et al. [134] studied the nature of JavaScript redirection spam techniques and demonstrated that obfuscated JavaScript is very prevalent to evade detection from anti-spam systems. Webb et al. [135] proposed an automated framework to identify web spam from emails spam based on redirection data. Fette et al. [136] attempted to detect phishing emails using several features which include whether a shortened link is embedded.

Other work focuses on the phishing detection on a specific area such as Twitter using the information of the account along with features of the tweets including redirection [50, 137, 138, 139, 131].

Redirection is also implemented in other areas to impact users' security and privacy. In the cross-site tracking field, people use redirection techniques to build user's visit activity [140]. Others employ redirection to bypass third-party cookie policies because the policies often do not consider redirection as cross-site requests [141]. Additionally, attackers use redirection to conduct affiliate marketing fraud [142]. But methodologies mitigating such fraud can employ redirection as an important feature [142, 143]. The significance of the redirection feature in affiliate marketing abuse detection and user privacy protection inspires us to take advantage of the redirection attributes to mitigate emerging phishing attacks.

4.10 Conclusion

Demonstrating the classic attacker-defender asymmetry, phishers carefully craft campaigns to circumvent anti-phishing defenses by capitalizing on cracks in defenders' understanding of the phishing landscape. Through the first in-depth analysis of redirection chains

in phishing websites, we have shown that these redirection chains are actively and effectively being leveraged by phishing campaigns to evade detection, and we have proposed practical approaches for enhancing anti-phishing systems to disrupt such chains. We believe that analysis systems such as PHISHPATH are critical to maintaining an understanding of phishers' evolving tactics. The anti-phishing ecosystem can employ our methodology to more expeditiously and more reliably detect sophisticated phishing.

Chapter 5

SPARTACUS: CLOAK USERS AGAINST CLOAKED PHISHING WEBSITES

After the analyses of cloaking and redirection techniques in phishing attacks that help evade anti-phishing mechanisms, I realize that phishers can continuously develop and deploy novel evasion techniques to the malicious websites. It indicates that the attackers are always one step ahead of the anti-phishing ecosystem. To protect the ecosystem proactively, I mitigate phishing attacks from the perspective of prevention, by completely neutralizing advanced phishing websites with fingerprinting-based cloaking techniques, and hence present Spartacus.

5.1 Prevalence of Fingerprinting-based Cloaking

Oest et al. [4] analyzed *.htaccess* files in phishing kits and demonstrated that fingerprinting-based cloaking is popular. To examine the prevalence of fingerprinting-based cloaking techniques used in advanced phishing kits, we manually inspect a random 10.93% (56) of phishing kits from a dataset by *phishunt.io* [110], and extract common patterns of fingerprinting-based cloaking techniques.

The patterns include (1) blocked words, for example, any potential crawler identification such as "google," "facebook," or "phishtank" (e.g., `blocked_words` in Figure 5.6), (2) IP checks that block visit from IP addresses, such as `bannedIP` in Figure 5.6, and (3) error responses (i.e., returning an error status code and an error web page). These attributes reflect the implementation of fingerprinting-based cloaking techniques in real-world phishing kits.

We use the identified patterns to automatically find fingerprinting-based cloaking techniques in phishing kits. Among the inspected kits in *phishunt.io* [110] from May 2020 to

Sensitive Word	Amount	Sensitive Word	Amount
bot	1,273	crawler	371
curl	581	facebook	223
google	447	phishtank	200
amazonaws	446	atn	125
compatible	432	spinner	113

Table 5.1: Top 10 Sensitive Words Appeared in the Phishing Kits Examined by Us.

July 2021, 410 of 512 contain fingerprinting-based cloaking techniques through IP, Referrer, or User-Agent. We also analyze 2,421 phishing kits provided to us by Cisco and find that all of these phishing kitsimplement fingerprinting-based cloaking techniques. Among them, 1,983 of the phishing kits contain a User-Agent check, and 1,660 contain an IP address check. In total, 96.52% (2,831) out of 2,933 phishing kits contain fingerprinting-based cloaking techniques.

Popular fingerprinting words. To understand the popularity of blocked User-Agent words (e.g., "bot" and "curl") that phishing kits use to evade anti-phishing crawler, we counted the appearance of 407 unique words in the User-Agent checks of phishing kits. Table 5.1 displays the top 10 blocked words. Note that one phishing kit can contain multiple rules using a word. The result shows that if a request contains "bot" or "curl" in the HTTP User-Agent header, phishers will return an error rather than phishing content. A potential victim's HTTP request, however, does not include those blocked words [144]. Based on this analysis, we can use the frequently blocked words and turn them into *trigger words* that will cause an HTTP request to evade the phishing content by triggering the phishing web site's fingerprinting-based cloaking.

5.2 Design

By turning fingerprinting-based cloaking against phishing web sites, we can create a proactive anti-phishing defense for users who attempt to visit such web sites. To this end,

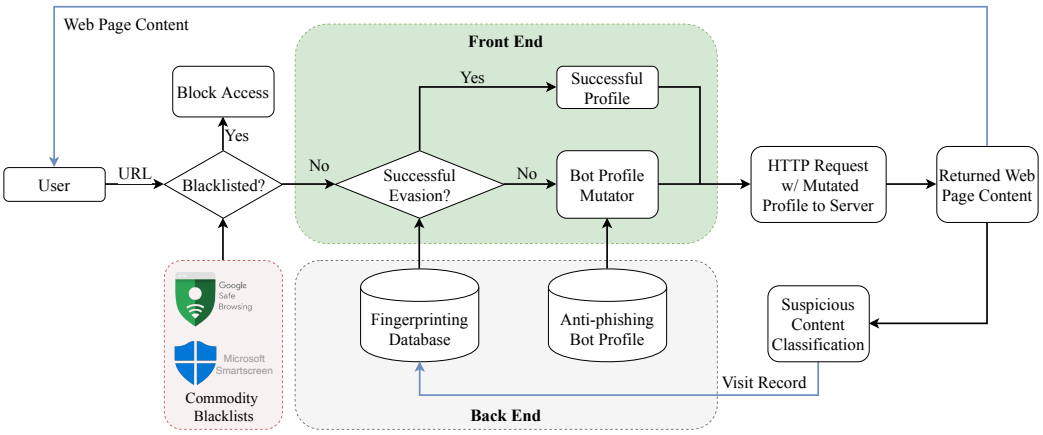


Figure 5.1: Spartacus Architecture and Its Workflow.

we design, implement, and evaluate Spartacus, a framework that automatically and selectively mutates HTTP requests made by a user's web browser to resemble those of an anti-phishing entity. If a user protected by Spartacus was to visit a phishing web site with fingerprinting-based cloaking, the mutated request would trigger the cloaking, which will then display benign content rather than a phishing page.

First, we define a *profile* as the set of fingerprintable attributes and their values. For instance, one profile might contain a User-Agent string with `bot`, an empty Referrer, or an AWS IP address. Profiles are used by Spartacus to generate an appropriate HTTP request for the target web site. Whenever phishers add new fingerprints to fingerprinting-based cloaking, these attributes can also be added to the Spartacus profiles.

5.2.1 Overview

Figure 5.1 illustrates the Spartacus architecture. The framework consists of two parts, the *front end* and the *back end*. The front end is responsible for the primary functionality, such as deciding if and how to mutate profiles, and the back end stores information.

When a user visits a URL, it is first checked against blacklists maintained by current anti-phishing systems, and if the URL is found, Spartacus outright *blocks* the access (by

displaying a prominent phishing warning). Otherwise, the front end queries the *Fingerprinting Database* to see if it has processed the URL before. If the URL is found, and was previously successfully mitigated, Spartacus will use the same profile to request the web site.

If the URL is neither blacklisted nor in the database, Spartacus will mutate the profile based on the *Anti-phishing Bot Profile Database*, skipping any mutations that previously failed to trigger cloaking [1] . Then, the HTTP request, with the mutated profile, is sent to the server. After receiving the page content, we determine whether it has suspicious content by using a classification engine that executes concurrently with Spartacus and runs in the background to avoid delaying page rendering. The purpose of the suspicious content classification is to verify if the mutated profile was effective at triggering cloaking. The Fingerprinting Database is, therefore, updated with the visited URL, the mutated profile Spartacus used, and the classification result.

5.2.2 Visit Pre-filters

When a user visits a URL, it must first pass through two pre-filters.

Blacklist Filter. With the contributions of the anti-phishing ecosystem, Spartacus can filter known phishing URLs that have already been blacklisted by commodity blacklists, specifically Google Safe Browsing and Microsoft SmartScreen. Any match will block access to the URL without further action.

Prior Visits. If the URL is not blacklisted, Spartacus will examine if the URL was previously visited by querying the Fingerprinting Database. If the URL was already visited and was successfully evaded, then Spartacus will use the successful profile mutation.

[1] We populate the Anti-phishing Bot Profile Database with known crawler-looking profiles (e.g., extracted from phishing kits as discussed in Section 5.1).

5.2.3 Bot Profile Mutator

The Bot Profile Mutator is responsible for profile mutation to trigger fingerprinting-based cloaking in advanced phishing web sites. Items in the profile can be modified or changed to camouflage the user as anti-phishing crawlers, for instance the User-Agent HTTP header. Generally, anti-phishing crawlers contain "bot" and "crawler," or the name of the company such as "Google" and "Facebook" in the User-Agent HTTP header. The mutator uses the *trigger words* that we automatically extracted from phishing kits (discussed in Table 5.1).

Another aspect is the Referrer HTTP header. Typically, the potential victim visits from the phisher's phishing lures. Therefore, phishers can block all visits that are not from the phishing lures.

Optionally, the profile mutator can leverage proxy servers to camouflage the user's IP address. For example, a proxy server on *AWS EC2* is useful because phishers have inferred that some anti-phishing crawlers use *AWS EC2* (according to our analysis in Table 5.1). In this case, the bot mutator can proxy the request through an evaded IP. Even though the proxy server can help evade fingerprinting-based cloaked phishing websites, it can also raise privacy concerns (discussed in Section 5.3). Therefore, users must consent to the privacy implications before enabling it.

In the mutation process, Spartacus appends one trigger word from the Anti-phishing Bot Profile Database to the user's own User-Agent string, following the order of the popularity in Table 5.1. Spartacus also avoids using trigger words that were not successful for the same URL. Additionally, Spartacus sets the Referrer to `None` (to remove the header). As for the optional IP/Hostname mutation, Spartacus reroutes the request to a proxy server, whose IP is in one of the most popular blocked IP ranges.

5.2.4 Suspicious Content Classification

After submitting the HTTP request to the server, Spartacus will receive an HTTP response from the server. After the suspicious content classification, there are four different possibilities:

1. The server is benign and responds with benign content.
2. The server is benign and responds with suspicious content.
3. The server is malicious and responds with benign content (e.g., error page or redirection to a benign web site, as shown in Figure 5.2c).
4. The server is malicious and responds with suspicious content.

For case (1), there is no security risk to the user, so we consider it a *successful* evasion. However, there is a possibility that the benign web site serves different content to Spartacus (due to User-Agent sniffing or other server-side techniques) and that Spartacus breaks the functionality of the web site for the user. Our experimental results (presented in Section 5.4.5) demonstrate that very few modern web sites change functionality/responses based on our changes to the HTTP request. In addition, we present in Section 5.4.5 a proposal, with experimental results, of adding another pre-filter to Spartacus to apply Spartacus to only suspicious web sites.

Case (2) happens when the user visits benign web sites that contain phishing-like features such as a login form, sensitive (phishy) words (e.g., "username" and "password"), and a submission button. As these are often indistinguishable from phishing pages (in fact, they are copied by phishers), we still consider it as an *unsuccessful* evasion. In the suspicious content classification, we still determine it as "suspicious" and the mutation will be marked as "unsuccessful." Similar to case (1), this situation does not affect users' browsing

activities on those web sites, according to the experimental result in Section 5.4.5. When the user visits the same web site next time, Spartacus will mutate the profile with other available trigger words in the Anti-phishing Bot Profile Database. Based on the evaluation result in Section 5.4.5, changing trigger words does not impact the web site's accessibility, layout, or functionality.

In case (3), the server is malicious with evasion techniques and determines that the visit from Spartacus is an anti-phishing bot visit. So it either returns an error web page, or redirects the visit to a benign web site. Consequently, Spartacus *successfully* prevents users from seeing phishing content, by triggering the fingerprinting-based cloaking techniques in the phishing web sites.

In case (4), the phishing web site either (a) does not perform any fingerprinting-based cloaking or (b) the profile failed to trigger the fingerprinting-based cloaking. In the former case (a), Spartacus cannot trigger any cloaking behavior in the phishing site, so we consider it as an *unsuccessful* evasion. However, our results in Section 5.5 demonstrate that these phishing sites are quickly detected by the anti-phishing ecosystem (median detection of 28 minutes). In the latter case (b), Spartacus will store the failed profile to help inform future visits to this URL to trigger the fingerprinting-based cloaking. Note that, from the perspective of the browser, this case is the same as case (2), which is why we cannot simply block the URL (we do not know if the server is malicious or benign).

5.3 Privacy

The Spartacus framework, as discussed in Section 5.2, is a client-side framework that runs solely on the user's machine. However, we strive to protect as much of the user's privacy as possible. In addition, these privacy protections are important to enable a potential centralized Spartacus in future work (discussed in Section 5.6).

5.3.1 Privacy Information

Spartacus requires four types of sensitive information from users: (1) *visited URL*: When Spartacus mutates the profile, it operates on a hashed URL, and only stores the hashed URL, along with the successfulness of evasion. (2) *The HTTP profile used*: Spartacus stores the user's HTTP profile to modify it. The privacy information in the profile includes the User-Agent string, which contains browser version, browser type, and operating system information, and the Referrer. (3) *Returned HTTP response*: Spartacus analyzes, but does not store, the returned HTTP response to inspect whether the HTTP response is suspicious. If valid content is returned, an external classification process will determine its suspiciousness by searching for content such as sensitive words and login forms [29]. The classification result is then stored to mark if the corresponding profile mutation successfully evades suspicious content. (4) *Potential proxy server monitoring*: if a user wants improved evasion performance beyond the User-Agent/Referrer mutation, he/she can choose to turn on the proxy server option in Spartacus. In this case, all user's HTTP requests will be sent through a proxy server, but the user can be susceptible to monitoring.

5.3.2 Privacy Consent and Protection

We ensure that Spartacus reasonably considers users' privacy, so we implemented both consent and protection methodologies to notify users of the data that is collected and prevent their information from being stolen and abused.

Consent. To ensure that users are aware of the privacy information Spartacus access and stores, a privacy policy consent notice is presented on first use. First, the privacy information used by Spartacus is summarized. Then, using Privacy Policies [145], a privacy policy is created for Spartacus. The information Spartacus collects and how it uses the information are described on the privacy policy page. Users can choose to opt out and not install

Spartacus if they do not consent. Especially, when users want to turn on the proxy server option to improve evasion performance, they must understand and consent to the option's trade-off. We created a dedicated privacy policy page for the proxy server option.

Protection Methodology. The privacy information Spartacus collects does not contain any PII, which minimizes potential harm. Spartacus stores the hashed URL, bot profile (IP address, User-Agent triggering word, and Referrer), and evasion success.

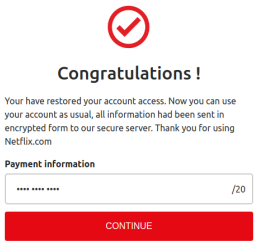
5.4 Evaluation

We implemented Spartacus as a Chrome browser extension, and we evaluate Spartacus through three perspectives: effectiveness, latency, and functionality impact. These three aspects demonstrate the feasibility of the Spartacus framework in practice because it can successfully evade advanced phishing web sites, introduce negligible latency to user browsing, and not introduce breakage.

Dataset. In our evaluation, we used two different datasets, a malicious one, to test the effectiveness of Spartacus, and a benign one, to understand its potential impact on benign web sites.

(1) *APWG Dataset*: For the effectiveness evaluation, Spartacus visited 160,728 live phishing web sites from November 2020 to July 2021 using the Anti-Phishing Working Group (APWG) URL feed [146], which is a curated dataset of reported phishing URLs, supported by a large number of collaborating members. Additionally, we leveraged another 8,474 live phishing web sites in the APWG Dataset to evaluate the effectiveness of IP mutation.

(2) *Benign Dataset*: To evaluate the impact on benign web sites, we collected a dataset of 60,848 benign domains, randomly selected from 629,843 domains in the Alexa Top One Million Domain List [147].



(a) Default browser visit.

(b) Spartacus browser visit with error.

(c) Spartacus browser visit with content.

Figure 5.2: Web Page Content from Default and Spartacus Browser Visits for a Cloaked Phishing Web Site.

5.4.1 Effectiveness

We evaluate the effectiveness of Spartacus by visiting the same phishing web sites with two different browser configurations: one with default settings (default browser) and the other with Spartacus installed (Spartacus browser). The phishing URLs for both visits are from the APWG Dataset. To reduce the impact of the selection of trigger words on the results, for each URL the Spartacus browser uses a profile with a random trigger word from the 407 trigger words, no Referrer header, and no IP proxy. Note that in Section 5.4.2 we evaluate the effectiveness of each trigger word and in Section 5.4.3 we evaluate the effectiveness of proxying the IP address. For each visit, we record the final landing web page content and URL.

In the experiment on 160,728 phishing URLs from APWG from November 2020 to July 2021, 132,247 (82.28%) did not contain malicious content in Spartacus. We consider an HTTP response from the Spartacus browser benign if its web page (1) is different from the one shown on the default browser and (2) does not contain suspicious content, such as "phishy" words or bad forms, according to the content features in our reimplementation of CANTINA+ [29].

Figure 5.2 demonstrates the difference of response web page content between the default and the Spartacus browser visit for a cloaked phishing web site. The content in Figure 5.2a shows the phishing content when the default browser visits it. When Spartacus mutates the HTTP profile to include a random trigger word and removes the Referrer header, the phishing web site shows the error web page in Figure 5.2b. Other phishing web sites redirect visitors to a benign URL instead of returning an error page. In this way, Spartacus receives the web page content shown in Figure 5.2c, also indicating a successful evasion.

This result shows that Spartacus can camouflage users as anti-phishing entities and prevent users from phishing content on phishing sites with fingerprinting-based cloaking techniques. Because the users see benign content (either an error page or a benign URL), the user is never exposed to the phishing attack, thus proactively preventing the user from falling victim to the phishing attack—even if they are the first user to visit the phishing URL.

5.4.2 Effectiveness of Trigger Words

In the prior Spartacus experiment, we used random trigger words for each URL visit. While this limited the impact of trigger word selection on the results, in our design of Spartacus the profile mutator selects trigger words in order of their popularity. Therefore, we evaluate the effectiveness of each trigger word on actually triggering fingerprinting-based cloaking techniques.

We tested all the trigger words by visiting each phishing web site with a different profile consisting of the trigger word, no Referrer header, and no IP proxy. Similar to the effectiveness evaluation, we also visit the web site using a default browser as a comparison. We conduct the experiment on 916 phishing web sites. 725 of them show web page differently at least under one trigger word between Spartacus browser and default browser.

In the 725 cloaked phishing web sites, each trigger word has different evasion capabil-

Trigger Word	Count	Trigger Word	Count
bot	720	atn	717
amazonaws	718	curl	717
phishtank	718	facebook	716
dwcp	717	crawler	716
google	717	katipo	713

Table 5.2: Top 10 Trigger Words That Evaded Phishing Content.

ities. Table 5.2 is the result of the top 10 trigger words in successfully evading phishing content. The word *bot* has the most sites evasion. 99.31% of the cloaked phishing web sites can be evaded by appending *bot* in the User-Agent. Compared with the popularity rank in Table 5.1, it is also the most popular blocked word in the phishing kits we examined. The effectiveness of this word in practice confirms its popularity in the phishing kits. Similarly, the top trigger words such as "amazonaws," "phishtank," and "google" also have high usage in phishing kits. An interesting note is that "bot" and "amazonaws" combined can trigger cloaking in all phishing web sites that used fingerprint-based cloaking techniques.

This result shows that trigger words can effectively evade phishing web sites with fingerprinting-based cloaking techniques. Furthermore, a small number of trigger words can evade a myriad of cloaked phishing web sites.

5.4.3 Effectiveness of Proxy Server Option

Even though Spartacus can successfully evade phishing content in over 80% of the phishing web sites by mutating User-Agent and Referrer headers, we want to analyze the effectiveness of mutating the IP address. Therefore, we conducted another experiment with a Spartacus profile that had default User-Agent header, has Referrer header, but proxied the connection through a server with an Amazon AWS IP address. Since the proxy server option may introduce privacy implications, we turn off this option by default. Users can

opt to use this feature in Spartacus only if they read, understand, and consent to the privacy implications.

In this experiment we used 8,474 phishing web sites. Similar to the evaluation procedure in Section 5.4.1, we visited those web sites in both the default and the Spartacus browsers (only changing the IP address according to the profile). Then, we compared the web page contents of each phishing web site on both visits and detected if the web page of the Spartacus browser does not contain suspicious content. Among the visited phishing web sites, Spartacus evaded 88.98% (7,540) of them through the proxy server. Therefore, Spartacus can evade phishing web sites that implement IP, User-Agent, or Referrer cloaking. Phishers may design emerging cloaking techniques in the future, however Spartacus was designed as an extensible framework so that fingerprint features can be added.

5.4.4 Efficiency and Latency

We next explore Spartacus' impact on the user experience when they visit benign web sites. By design Spartacus may introduce latency to the HTTP request, due to the database query, HTTP profile mutation, and returned content inspection.

We conducted an experiment to measure the latency of Spartacus from the following three perspectives: database query, profile mutation, and content inspection. We used *exthouse* [**?**], which analyzes the impact of a browser extension on web performance, as our test bench, which contains five major measurements:

- Time to Interactive (TTI): the time it takes for the page to become fully interactive with the extension.

- First Input Delay (FID Δ): the time from when a user first interacts with the web site to the time when the browser is actually able to begin processing event handlers in response to that interaction.

- Scripting Time (Scripting Δ): the amount of time JavaScript execution in the extension.

- Long Task (Added Long Task): this value represents a sum of Long Tasks added by the extension, where Long Tasks are defined as a task that blocks the main thread for 50 ms or more [2] .

- Extra CPU Consumption (Extra CPU Time): the extra CPU consumption of the extension for each URL the browser visits.

The lower the factors are, the better the web site performs with the tested extension. Lastly, *exthouse* creates a score for the extension. A higher score reflects a better performance of the extension.

Table 5.6 illustrates the *exthouse* scores of the top 10 Chrome extensions [**?**] and Spartacus when visiting benign and malicious web sites. We tested these extensions with 100 web sites, including half benign and half malicious, and used the average in the metrics. Spartacus has a score of 100, based on a 20 ms FID, 0 scripting delta, and 800 ms of TTI when visiting benign web sites. The metrics of Spartacus visiting malicious web sites also outscores those of other popular extensions. Even though it takes a longer time to interact with the malicious web site, it is still acceptable because Spartacus needs time to mutate the profile, which is still less time than other extensions. For instance `Avira Browser Safety` (ABS) is an extension that warns users if the web site is unsafe. It adds long tasks and extra CPU time when visiting malicious web sites.

This evaluation result shows that Spartacus adds minimal overhead to web browsing. The inspection result shows that Spartacus outscores popular Chrome extensions and has negligible impact on the performance of the web sites, compared with other extensions.

[2] `https://developer.mozilla.org/en-US/docs/Web/API/Long_Tasks_API`

Experiment	Tested	Passed	Blocked	Different Layout
W/out algorithm 1	60,848	60,574	150	124
W/ algorithm 1	60,848	60,688	29	39

Table 5.3: Coarse-grained Experiment Result Of Spartacus with and Without Applying Logic of Mutating HTTP Profile.

5.4.5 Impact on Benign Web Sites

Merzdovnik et al. discovered that add-ons can cause some web sites to malfunction (e.g., they found that the *PrivacyBadger* extension led to a large number of timeouts and therefore to a potentially large number of malfunctioning web sites) [148]. Therefore, it is important for Spartacus to minimize the negative impacts on benign URL visits. Impacts may include the ability to access web sites, the correct display of web site layout, and the correct web site functionality. To evaluate functionality of benign web sites, we conducted two experiments: a Coarse-Grained (a large-scale evaluation with automated analysis) and Fine-Grained (a small-scale evaluation with in-depth manual analysis). In each experiment, for each URL, the Spartacus browser used a profile with a random trigger word, no Referrer header, and no IP proxy.

Coarse-Grained Experiment

In the Coarse-Grained experiment, we intended to evaluate if the Spartacus framework has negative impacts on access to the web site or the web site layout through automated analysis of results on a large crawl of benign domains. We randomly sampled 60,848 (9.66%) from the 629,843 URLs in Alexa Top One Million Domain List [147] and visited them in both default and Spartacus browsers. We compared the resulting web page screenshot and HTML similarity on the visited URLs. The result is shown in Table 5.3: 0.25% (150) have different layouts, and 0.20% (124) block access to the Spartacus browser.

We first manually examined the results to identify why the Spartacus browser shows different layouts from the default browser. We found that although the screenshot and HTML were dissimilar between the default and Spartacus browser visits, such differences did not impact the use of the web site. Figure 5.7 and Figure 5.8 show typical differences in browser rendering between the default and Spartacus browser visits. For example, the web page is rendered differently in terms of screenshot similarity between the default and Spartacus browser visits shown in Figure 5.7. The difference here is due to the shape of the buttons, different background color, and content spacing. In Figure 5.8, a window popped up to ask for permission to use cookies in the default browser, but did not in Spartacus's visit. The cookie request pop-up that was missing in the Spartacus browser is not due to the extension: in 10 visits in different default browsers, the pop-up appeared only three times.

Benign web sites with security mechanisms. Some benign web sites are built on web hosting services such as Cloudflare and Akamai, and the services contain security mechanisms such as anti-DDoS and anti-crawling. Therefore, to make sure that users can successfully visit those web sites with the protection of Spartacus, we visited 5,000 Cloudflare-based and 5,000 Akamai-based benign web sites using Spartacus. In total, we could successfully visit 99.86% of 10,000 web sites. 14 benign sites were inaccessible. It is mainly because the web site owners employ traffic filtering mechanisms over the CDNs. With such low false-evasion rate, users can visit most benign web sites hosted on the CDNs successfully, and can report the falsely evaded benign web sites to the Spartacus provider, who can asynchronously inspect them and force Spartacus to use the default profile to visit the web sites.

Potential Improvement. Although Spartacus has low false positives when visiting benign web sites, there is still the possibility to reduce false positives. We inspected the 150 benign web sites that were falsely evaded by Spartacus and 150 cloaked phishing web sites that were successfully evaded by Spartacus. To further differentiate the two categories, we

extracted domain reputation [149], domain age [150], and top viewed sub-domains [151] of URLs from benign and malicious web sites. Cisco Talos [3] defines reputation of a domain using five categories: Trusted, Favorable, Neutral, Questionable, and Untrusted [149]. Reputation-wise, 136 out of 150 phishing URLs have a reputation lower or equal to Neutral level, the lowest of which is Untrusted. In contrast, only 24 of 150 benign URLs have a lower reputation than Favorable, the lowest of which is Questionable (and only one is Questionable). In terms of domain lifespan, the mean value of domain duration since registration for benign URLs is 4,692 days and the median is 4,521 days. However, the average lifespan of the malicious domains is 1,618 days, with a median of 900 days. Moreover, all 150 benign URLs fall into the top viewed sub-domains of the corresponding domain names, while none of the phishing ones matches.

Therefore, we summarize that a legitimate domain has a higher reputation and longer life than a malicious one, and they are within top viewed sub-domains. We can further reduce the possibility of Spartacus falsely evading benign web sites by introducing another pre-filter to Spartacus before mutating the HTTP profile.

We choose the phishing domain age that resides on 75% in the list (1,501) and the Neutral level as thresholds because these thresholds clearly divided trustworthy domains from un-trustworthy domains. With this pre-filter, if a URL has a lifespan lower than 1,501 days and its reputation level is Neutral or worse, or its sub-domain is not top viewed, then Spartacus will mutate the HTTP profile. The logic is shown in Algorithm 1.

We evaluated this augmented version of Spartacus and found that 29 legitimate domains show different web page content on the default and augmented version of Spartacus browsers, as listed in Table 5.3. Hence, only 0.04% of 60,848 domains result in a false positive detection of a phishing web site. This is because the 29 domains do not meet the trustworthiness requirements and therefore Spartacus mutated the profile when visit-

[3] Cisco Talos is a threat intelligence service and used by other studies [152, 153, 154].

Algorithm 1: Logic of mutating HTTP profile

1: $p = default\_profile$
2: $u = url\_to\_visit$
3: **if**
4: ($reputation(u) \leq Neutral$) `and`
5: ($duration(u) \geq 1,501$) `or`
6: ($u.domain$ NOT in top reviewed sub-domains) **then**
7: $\quad p = mutate\_http\_profile(p)$
8: **end if**
9: $send\_request(p)$

Evaluation Perspective		Amount	
		Default	Spartacus
Accessibility		58	58
Correct Layout		58	58
Proper Functionality	Click Buttons	58	58
	Online Chat	3	3
	Shopping Cart Add	5	5
	Registration/Login	22/22	22/22

Table 5.4: Fine-grained Experiment Result Of Spartacus Visit, Compared with the Result of Default Browser Visit.

ing them. As one possible mitigation, we can provide a channel for users to report falsely evaded web sites. After receiving a report, we can conduct a manual inspection and force Spartacus to trust the false-positives. In comparison, phishing URLs can still be evaded through Spartacus because they pass through the augmented pre-filter.

Fine-Grained Experiment

Inspired by the methodology used by Snyder et al. [155] and Trickel et al. [156], in the Fine-Grained experiment, we aim to manually evaluate the operation of web sites visited through Spartacus.

This methodology concentrates on the operation of a web site from the perspective of

the user. Even though Spartacus may introduce an error to a web site, if the users do not perceive any difference when browsing, then we consider that Spartacus does not negatively impact the web site. This methodology focuses on user-facing impacts to benign web site functionality, and the experiment was performed by the authors manually.

The experiment includes the evaluation of a web site's rendering and interactions between visitors and the web site. There are four steps in the experiment methodology: (1) Open legitimate domains in a browser with Spartacus installed and also in a browser with default settings. (2) Inspect the accessibility of the web site, similar to the Coarse-Grained experiment. (3) After successful web page content retrieval, compare the layouts between different visits. (4) Interact with links, buttons, and other activities such as register/login, online chat, or shopping to make sure that the web site performs correctly. (5) Finally, test the authentication functionality to ensure that Spartacus will not impact it.

We randomly selected 60 domains from the Alexa Top One Million List, choosing 20 every 200,000 (for an even distribution), and the result is displayed in Table 5.4. As a comparison, the default browser had the same result as that of Spartacus. Among the 60 legitimate domains, 58 were accessible. Two domains were inaccessible even in the default browser, so we suspect that they are offline. For the 58 accessible domains, we followed the steps described previously to inspect them. All of them have the same layout as the visit from the default browser. Then, we interacted with the 58 web sites by clicking buttons, chatting online, and adding items into the cart if it is a shopping web site. All 58 web sites performed normally. Lastly, we registered an account on 15 web sites, and all were successful in Spartacus. Even though we can successfully register an account, we still need to make sure that we can log in properly with those accounts to test the authentication process under Spartacus. The results shows that all the accounts we registered during the Fine-Grained experiment could be logged in successfully.

With the experimental results from both the Coarse- and Fine-Grained experiments, we

can summarize that Spartacus has little impact on the accessibility and visibility on benign web sites. Therefore, Spartacus can protect users from visiting advanced phishing web sites while keeping their normal browsing activities.

Trigger words on benign web sites. When browsing benign web sites, Spartacus can think that the visit resulted in evasion failure, and then mutate future visits with new trigger words. Therefore, we conducted an experiment to show that these trigger words do not impact the layout and functionality of benign web sites. We tested the top 10 popular trigger words in Table 5.1 on 30 randomly selected benign domains. We then compared the screenshot and HTML similarity between the default browser visit and 10 Spartacus browser visits using the different trigger words. The result shows that each of the 30 benign domains have the same layout using different trigger words. Additionally, we manually tested the functionality as in the Fine-Grained experiments and found that all web sites performed correctly with different trigger words.

Benign web sites with risk-based authentication mechanism. Risk-based authentication (RBA) mechanism prevents web sites from requiring users to use Two-Factor Authentication by inspecting the features in an HTTP request such as IP addresses and/or User-Agent string [157]. As major web sites such as Amazon, Google, LinkedIn, and Facebook have employed such a mechanism [157], we evaluate Spartacus's ability to trigger Two-Factor Authentication in RBA enhanced web sites. To this end, we visited eight web sites that are known to employ RBA using Spartacus [4] . These web sites cover all three types of RBA implementations mentioned in prior work [157]. The result shows that we could successfully visit all eight web sites without encountering a Two-Factor Authentication prompt, which demonstrates that Spartacus does not cause inconvenience for users visiting RBA protected benign web sites.

[4] Amazon, Facebook, GOG.com, Google, iCloud, LinkedIn, Steam, and Twitch [157].

Long-term Use

To determine long-term impact on the user experience, we evaluate how Spartacus performs in a long term usage scenario. The authors installed Spartacus in their primary browsers for daily use for a period of one month. During the experiment, we looked for abnormalities, such as unexpected access blocking, frequent risk-based authentications (e.g., re-CAPTCHA and two-factor authentication), and slow page rendering. After the one-month experiment, the authors did not encounter with any abnormal actions during normal browsing.

Reason of Low Breakage.

In both Fine-Grained and Coarse-grained experiments, the Spartacus browser could successfully request and render benign web sites and allow users to interact with them as usual. It is mainly because we only append one of the trigger words in the User-Agent string, instead of replacing the string with a crawler one. As we discovered in Section 5.1, phishing servers will deny the request because they employ an aggressive filtering mechanism—blocking access as long as there is any suspicious patterns in the User-Agent string. However, benign web sites perform anti-crawling in a different way. For example, they monitor new or existing user accounts with high levels of activity and no purchases, or they detect abnormally high volumes of product views as a sign of non-human activity [158]. Additionally, benign web apps such as *WordPress* [159] check the User-Agent string mainly because they need the visitor's browser version to deliver the best web page layout and according functionalities. Therefore, the difference between benign and malicious server's anti-crawling mechanisms allows Spartacus to evade phishing sites and access benign ones.

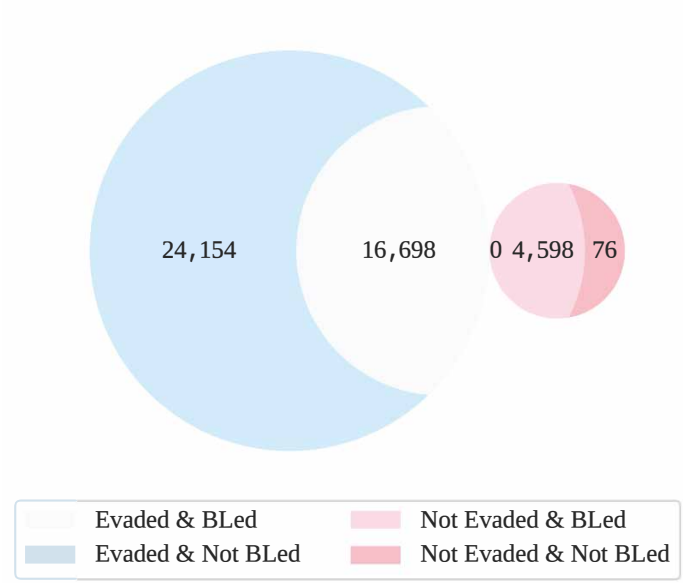


Figure 5.3: Venn Diagram Describing the Ability of Spartacus and Current Anti-phishing Systems Against Phishing Web Sites. The Unit Is the Amount of Phishing Web Sites.

5.5 Ecosystem Support

Note that in our experiments (detailed in Section 5.4.1) Spartacus cannot evade 17.72% of the URLs. We hypothesize that these phishing web sites do not rely on fingerprinting-based cloaking techniques. Nowadays, the anti-phishing ecosystem such as Google Safe Browsing and VirusTotal can quickly detect and/or blacklist such phishing attacks. Therefore, we can rely on support from the ecosystem to handle the phishing web sites that Spartacus cannot evade.

To verify our hypothesis, we evaluated the blacklist/detection speed of current anti-phishing systems on the examined phishing URLs. We submitted the phishing URLs to Google Safe Browsing and VirusTotal at the same time that Spartacus visited them, and queried the results every 15 minutes to calculate the speed. Due to the deployment time of this experiment, we submitted 45,526 phishing URLs: 40,852 that Spartacus could evade and 4,674 that Spartacus could not evade. Among the 4,674 submitted phishing URLs that

Spartacus could not evade, Google Safe Browsing and VirusTotal combined blacklisted 4,598 of them. The remaining 76, after manual inspection, were found that they were not phishing web sites and were falsely reported to APWG. This evaluation result verifies that the ecosystem currently can protect users from phishing web sites that Spartacus cannot evade, which acts as a complement to Spartacus. In contrast, we submitted 40,852 phishing web sites that Spartacus successfully evaded. This result shows that 24,154 of them were not detected or blacklisted by the anti-phishing systems.

Figure 5.3 overviews the ability of Spartacus along with support from the anti-phishing ecosystem. Within our dataset, advanced phishing web sites significantly outnumber basic ones, which is shown in Figure 5.3 as the blue circle and red circle, respectively. However, the anti-phishing ecosystem detects only 40.87% of the submitted evaded phishing URLs. As a comparison, Spartacus can evade 89.73% of submitted phishing URLs. The ecosystem can mostly handle the non-evaded phishing URLs as a complement to Spartacus, and hence both advanced and basic phishing web sites can be evaded or detected.

We also measure the detection speed of current anti-phishing systems, which is visualized in Figure 5.4 (within 24 hours) and Figure 5.5 (whole frame). All submitted phishing web sites that Spartacus cannot evade are detected/blacklisted in two hours, and 50% of these can be detected within 22 minutes. As a comparison, current anti-phishing systems do not perform well against phishing web sites that can be evaded by Spartacus. The median detection time is 154 minutes. However, within 24 hours, they only detect/blacklist 76.16% of the cloaked phishing web sites. Moreover, it can take as long as 47.82 hours to finally detect a cloaked phishing web site. This reflects the ability of the current anti-phishing ecosystem against phishing web sites: for basic phishing web sites, current anti-phishing systems can react and blacklist them quickly, however, for advanced phishing, it takes a long time to detect, which is exploited by phishers to victimize users. Spartacus, however, only needs an average of 3.2 seconds (based on Table 5.6) to evade advanced phishing

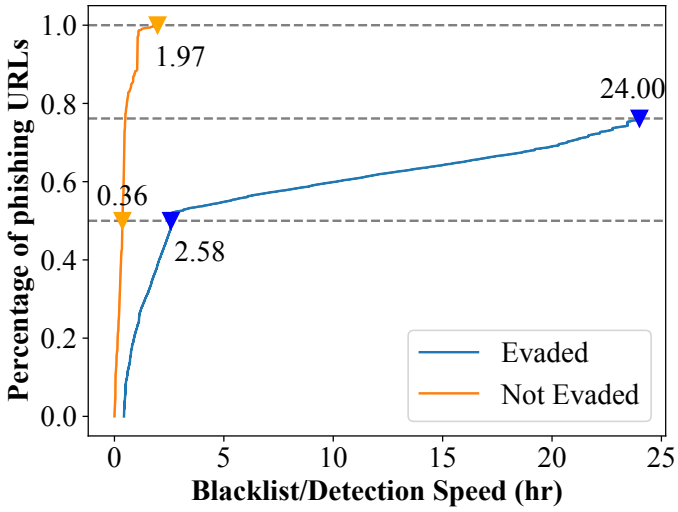


Figure 5.4: CDF of Blacklist/Detection Time Within *24 Hours* of Current Anti-phishing Systems Against Detected Phishing URLs Evaded and Not Evaded by Spartacus.

web sites. Therefore, Spartacus can not only greatly shorten the *golden hour* [160] left by the current anti-phishing ecosystem, but also evade cloaked phishing web sites that the ecosystem cannot detect.

5.6 Mitigating Server-side Cloaking

According to our observation and analysis in Section 5.1, phishers use fingerprinting-based cloaking techniques to accomplish their phishing attacks. We expect that the sophistication of phishing web sites will continue to improve, and that advanced phishing kits will create more fingerprints to inspect, match, and block requests that appear to be from anti-phishing entities.

Although researchers and organizations have proposed mitigations for phishing web sites [29, 161], they all require malicious web page content to feed the classifier and make decisions. Server-side cloaking techniques deny requests from the anti-phishing systems, and their methodologies may not be useful. To demonstrate this, we selected 500 cloaked

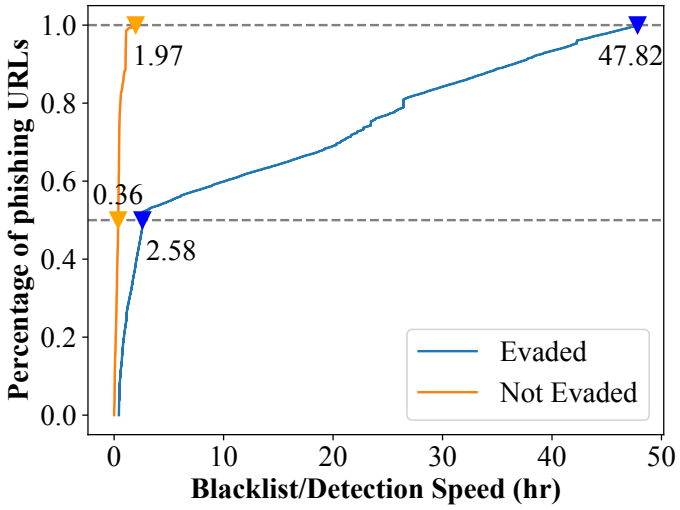


Figure 5.5: CDF of Blacklist/Detection Time of Current Anti-phishing Systems Against Detected Phishing URLs Evaded and Not Evaded by Spartacus.

Model	Precision (%)	Recall (%)	FPR (%)	FNR (%)	F1	ACC (%)	ROC-AUC	Proc Time (s)
PHISHPATH (regular)	92.28	92.26	11.48	7.74	0.92	90.76	0.95	35,985
PHISHPATH (w/ sub-chain)	86.78	92.28	15.76	7.72	0.89	88.49	0.94	**6,797**
CANTINA+ (regular)	85.84	89.18	19.57	10.82	0.87	85.42	0.95	23,133
CANTINA+ w/ PHISHPATH	**98.95**	**98.64**	**1.53**	**1.36**	**0.99**	**98.57**	**0.99**	59,118

Table 5.5: Evaluation Metrics of Evaluation of Our Classification on Websites with Redirection.

phishing web sites that can be evaded by Spartacus and 500 benign web sites from the Alexa Top One Million List to test our implementation of CANTINA+ [29], which is a phishing classifier with URL-, web-, and HTML-based features. The result, depicted in Table 5.5, shows a high false-negative rate when classifying cloaked phishing web sites. Therefore, the ecosystem should ensure that existing and new detection and mitigation systems are capable of adapting to fingerprinting-based cloaking techniques.

While Spartacus attempts to modify the user's HTTP requests to appear as anti-phishing systems, to best combat server-side cloaking anti-phishing systems should carefully modify the HTTP requests of their crawlers to mimic users. In this case, the anti-phishing systems can bypass the cloaking techniques and retrieve the actual malicious content. For example, they should avoid sending requests based on the IP addresses of well-known anti-phishing entities.

Additionally, Spartacus can be extended to share resources among users. Instead of only locally recording visited URLs and successfulness of profile mutations, Spartacus can merge the visit history from users in a centralized server. Then, the server can distribute and update periodically to the clients. In this way, users can benefit from an up-to-date Fingerprinting Database because Spartacus knows to block access if it ever successfully evaded the web site from other users. Furthermore, the Fingerprinting Database is designed to minimize privacy issues in this centralized setting, because all URLs are hashed and only the triggering words and proxy server IP are stored in the database.

5.7 Countermeasures to Spartacus

Even though Spartacus can successfully prevent users from seeing phishing content in fingerprinting-cloaked phishing web sites, attackers will explore countermeasures to mitigate it.

5.7.1 Using Other Cloaking Techniques

There are different types of phishing web sites in the wild, roughly categorized into basic and advanced. Within advanced phishing web sites, server-side and client-side cloaking techniques are two techniques that help evade detection from the anti-phishing ecosystem. Because Spartacus focuses on the evasion of phishing attacks with server-side cloaking techniques, attackers can use other types of cloaking techniques to harm individuals and

organizations.

Phishers can use basic phishing web sites, phishing web sites with client-side cloaking, or those with User Interaction Cloaking (e.g., CAPTCHAs). As we presented in Section 5.5 and with the analysis results from Oest et al. [162], basic phishing web sites can be quickly detected and blacklisted by anti-phishing systems in a median of 28 minutes. As for client-side cloaking techniques, phishers can implement them into their web sites to bypass Spartacus. However, Zhang et al. [108] proposed a methodology to detect such evasion by force-executing JavaScript. Hence, client-side cloaked phishing web sites can also be detected using prior techniques.

Finally, phishers can create a CAPTCHA web page as a barrier before showing phishing content. Such a technique can bypass evasion from Spartacus. However, using CAPTCHA may lower phishers' profit because it is time consuming and challenging for potential victims [163]. Secondly, it does not distinguish real users from anti-phishing crawlers because every visitor needs to solve the puzzle [163]. Recently, researchers have proposed methodologies of bypassing CAPTCHAs [164, 165, 108], which further allows anti-phishing crawlers to bypass them.

In a future with Spartacus deployed and support from the anti-phishing ecosystem, it is challenging to bypass all anti-phishing methodologies while allowing only potential victim traffic to visit. Such dilemmas force attackers to either spend resources inventing new evasion techniques or accept reduced profit.

5.7.2 Emerging Phishing Based on Spartacus

We assume that phishers can gain full knowledge of Spartacus and develop countermeasures accordingly.

For example, phishers could develop stateful server-side cloaking techniques: allowing the first person with a matching template to evade the phishing content, then change the

cloaking so that future visits would see phishing content. This could affect the second user who visits the same phishing web site, because Spartacus uses the "successful" profile for the user. Hence, the user views the phishing content due to the stateful nature of the server-side cloaking. However, by design, the suspicious content classification module is run externally and determines whether the returned web page has suspicious content, no matter if Spartacus mutates the profile or uses a successful one. Therefore, when the classification module determines that the profile is unsuccessful to evade phishing, Spartacus will mark it as failed and will select a new one in the future.

As another example, phishers could enumerate the bot profiles in Spartacus and develop high-fidelity cloaking techniques which identify anti-phishing ecosystem HTTP profiles more precisely. For example, phishers could only cloak visits that contain the exact User-Agent string of anti-phishing crawlers [5] . This technique could bypass Spartacus's evasion. However, these precise fingerprints increase the ease of anti-phishing systems to successfully bypass the cloaking by trivially changing their User-Agent string. Finally, phishers could develop complex and advanced fingerprinting techniques to use fingerprints that Spartacus does not consider, such as the order of HTTP headers, TCP/IP fingerprints, support for esoteric HTTP features (e.g., supporting the `100 Continue` response code), timing side-channels, and so on. While some of these fingerprints could be added to Spartacus, it might not be technically feasible to add all of them. Therefore, we could work with anti-phishing entities (or they could deploy Spartacus themselves) to integrate the exact Spartacus framework into their anti-phishing systems, so that these low-level fingerprints would be identical to Spartacus users.

[5] E.g., phishers only block visits whose User-Agent perfectly matches *Mozilla/5.0 (Linux; Android 5.0; SM-G920A) AppleWebKit (KHTML, like Gecko) Chrome Mobile Safari (compatible; AdsBot-Google-Mobile; +http://www.google.com/mobile/adsbot.html)*.

5.8 Limitations

Even though Spartacus can protect users from a diverse array of sophisticated phishing web sites using server-side cloaking techniques in the wild, our framework should be considered alongside certain limitations.

5.8.1 Spartacus Design

Phishing Classification. The Spartacus framework is not a phishing classification system. Instead, it camouflages users as security crawlers when they visit web sites with cloaking techniques and can evade malicious content if they use fingerprinting-based cloaking. This approach proactively protects users in nearly real time. As evaluated in Section 5.4, Spartacus can evade 82.28% of phishing web sites in real time using only User-Agent and Referrer mutation, with a negligible impact on benign web sites. Previous work has proposed methodologies classifying phishing web sites with high accuracy [41, 161]. Therefore, with Spartacus and existing classification methodologies, the anti-phishing ecosystem can cover a broader range of phishing attacks.

HTTP request mutation. As discussed in Chapter 2, fingerprinting-based cloaking techniques can inspect IP, Hostname, User-Agent, Referrer, and other fingerprints to classify whether the visitor is an anti-phishing crawler or a potential victim. In Spartacus's design, we consider mutating User-Agent and Referrer in the HTTP request, along with changing the IP address using proxy servers. There is a limitation where Spartacus cannot evade phishing web sites that only identify crawlers/bots by new fingerprints that Spartacus does not mutate. One solution for the potential limitation is that we intentionally designed Spartacus as an extendable framework. In this case, Spartacus can remain up-to-date to evade new cloaked phishing web sites.

5.8.2 Spartacus Deployment and Evaluation

Phishing kit analysis. In the analysis to understand the prevalence of fingerprinting-based cloaking, we hope to include as many phishing kits as possible. Due to resource limitations, we only analyzed phishing kits from *phishunt.io* [110] and those from the public dataset from Cisco. Within both resources, we analyzed 2,933 phishing kits and summarized that the fingerprinting-based cloaking techniques exist in 96.52% of the phishing kits. We believe that this analysis demonstrates the prevalence of fingerprinting-based cloaking techniques.

Data collection. We selected the APWG Dataset to evaluate the effectiveness of Spartacus. Due to infrastructure and resource limitations, we were only able to test Spartacus over total of nine months from November 2020 to July 2021. Even though additional data crawling would be desirable to evaluate Spartacus, the APWG Dataset provides a breadth of phishing data collection because it contains diverse types of phishing web sites targeting different brands. The phishing web sites are submitted periodically by collaborating members including anti-phishing systems and financial organizations impersonated by phishing web sites. Overall, we tested Spartacus on over 130,000 live phishing web sites and verified that Spartacus could evade malicious content by triggering fingerprinting-based cloaking. Therefore, we believe this limitation is mitigated to the extent allowed by current resources.

5.9 Related Work

Researchers have studied phishing for several decades. They have proposed several methodologies to detect phishing attacks based on features from the URL, content, etc., and then warn users before visiting the deceptive web sites. Some work analyzes the URL of a suspicious web site based on the lexical features or URL ranking to determine the maliciousness of the site [33, 166, 167, 168]. Others collect web page content and detect

phishing web sites with textual and visual similarity features [37, 38, 169]. Combining these techniques with other available features, including both URL and web page content, researchers developed blacklist-based anti-phishing systems such as Google Safe Browsing [41] to protect users from visiting suspicious web sites. All the proposed methodologies in the past, however, have a limitation that they are detection systems, and require certain features to classify the maliciousness, which can take time to acquire (due to cloaking). Furthermore, Bijmans et al. [170] found that the median uptime for phishing domains is 24 hours, showing the fast move of phishers. The delay of detection from anti-phishing systems and quick action of phishers extends the gap to mitigating phishing attacks [160].

With the large-scale implementation of cloaking techniques in phishing attacks [160, 162, 5, 4], researchers realize that sophisticated phishing attacks are responsible for a substantial portion of damage and that the whole ecosystem should prioritize mitigating phishing with evasion techniques. Cloaking techniques make anti-phishing more challenging because it becomes more and more difficult to retrieve the phishing content, which most anti-phishing systems depend on. With a very limited amount of web site features, current anti-phishing systems cannot precisely determine the maliciousness.

Therefore, analysis and detection of server-side and client-side cloaking techniques have been proposed to fight against such sophistication. For client-side cloaking techniques, Zhang et al. [108] proposed CrawlPhish to force-execute JavaScript snippets in the HTML response to reveal malicious content. As for server-side cloaking in phishing, previous work [45, 7, 4] categorizes types of server-side cloaking through analysis of compromised phishing kits.

We consider the nature and prevalence of cloaked phishing web sites [4, 5] and provide a novel methodology to proactively prevent users from seeing phishing content. Rather than bypass cloaking techniques in phishing web sites, Spartacus deliberately triggers them and hence prevents users. Our framework is also extensive with the ability to add fingerprints

that phishers use in the future.

5.10 Conclusion

Through our analysis of compromised phishing kits, we understand that fingerprinting-based cloaking techniques are largely implemented in the sophisticated phishing attacks and help to evade visits from anti-phishing entities. Such evasion is difficult to mitigate because phishers can always include new features of the up-to-date anti-phishing crawlers and identify them.

We consider this problem from a different perspective. The current state of the ecosystem is that the anti-phishing entities identify the fingerprints that phishing kits use to trigger cloaking and design new crawlers without those fingerprints, which are then learned by phishers. The phishers then add new fingerprints—in a never-ending cycle.

The Spartacus system proposes a new angle for the anti-phishing ecosystem to fight against cloaked phishing web sites. Rather than attempting to circumvent cloaking techniques, Spartacus beats the advanced phishing web sites at their own game by deliberately triggering the cloaking behavior to protect users against cloaked phishing web sites. For benign web sites, we demonstrated that Spartacus has negligible impact on users' access, web layout, and web functionality.

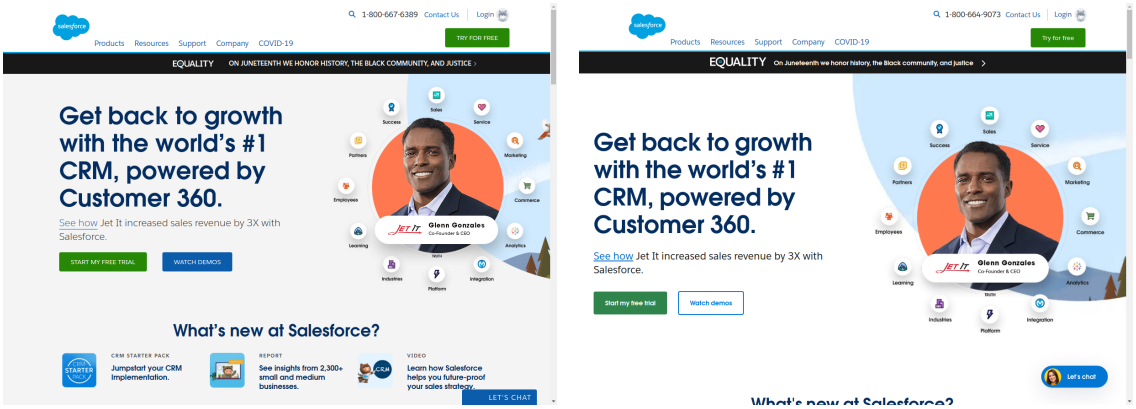
Due to the rise of sophisticated phishing web sites in the wild, we believe that automated evasion systems such as Spartacus are essential to keep trapping phishers in a lose-lose dilemma where they cannot differentiate real users from anti-phishing crawler visits. Methodologies such as ours can be incorporated into the ecosystem to more expeditiously and reliably evade sophisticated phishing, thus proactively protecting users from phishing attacks.

```php
$hostname = gethostbyaddr($_SERVER['REMOTE_ADDR']);
$blocked_words = array("above","google","softlayer","amazonaws","cyveillance",
"phishtank","dreamhost", "netpilot","calyxinstitute","tor-exit", "paypal", ...);
foreach($blocked_words as $word) {
    if (substr_count($hostname, $word) > 0) {
    header("HTTP/1.0 404 Not Found");
        die("<h1>404 Not Found</h1>
        The page that you have requested could not be found.");

    }
}
$bannedIP = array("^66.102.*.*", "^38.100.*.*", "^107.170.*.*", "^149.20.*.*",
"^64.233.160.*", "^72.14.192.*", "^66.102.*.*", "^64.18.*.*", "^194.52.68.*",
"^212.143.*.*", "^212.150.*.*", "^212.235.*.*", "^217.132.*.*", "^50.97.*.*",
"^217.132.*.*", "^209.85.*.*", "^66.205.64.*", ...);
if(in_array($_SERVER['REMOTE_ADDR'],$bannedIP)) {
     header('HTTP/1.0 404 Not Found');
     exit();
} else {
     foreach($bannedIP as $ip) {
          if(preg_match('/' . $ip . '/',$_SERVER['REMOTE_ADDR'])){
               header('HTTP/1.0 404 Not Found');
               die("<h1>404 Not Found</h1>
               The page that you have requested could not be found.");
          }
     }
}

if(strpos($_SERVER['HTTP_USER_AGENT'], 'google')
    or strpos($_SERVER['HTTP_USER_AGENT'], 'msnbot')
    or strpos($_SERVER['HTTP_USER_AGENT'], 'Yahoo! Slurp')
    or strpos($_SERVER['HTTP_USER_AGENT'], 'YahooSeeker')
    or strpos($_SERVER['HTTP_USER_AGENT'], 'Googlebot')
    or strpos($_SERVER['HTTP_USER_AGENT'], 'crawler')
    or strpos($_SERVER['HTTP_USER_AGENT'], 'facebookexternalhit') !== false
    or ...)
{ header('HTTP/1.0 404 Not Found'); exit; }
```

Figure 5.6: Simplified PHP Code Snippet of Fingerprinting-based Cloaking in a Phishing Kit, Checking IP, Hostname, and User-Agent.

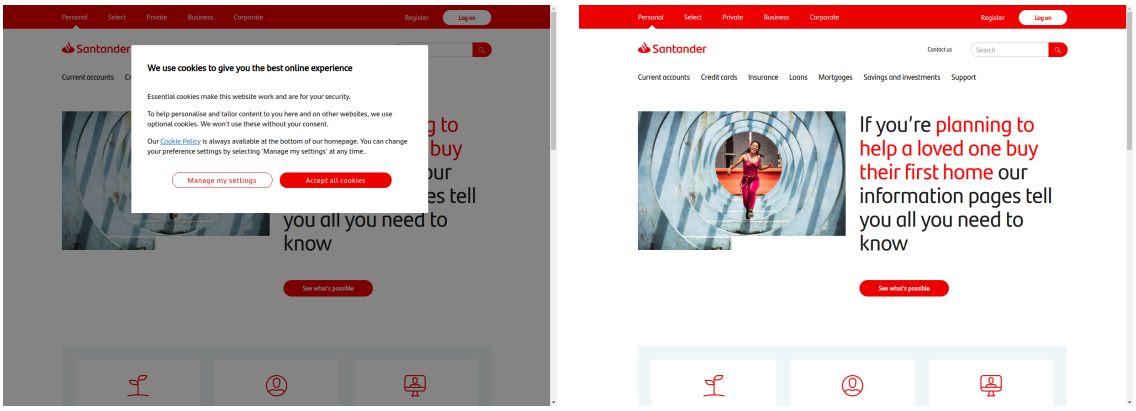
Name	**Score**		**FID Δ (ms)**		**Scripting Δ (ms)**		**TTI (ms)**		**Added Long Task (ms)**		**Extra CPU Time (ms)**	
	B	M	B	M	B	M	B	M	B	M	B	M
Grammarly for Chrome	50	60	20	190	300	1,000	1,300	3,500	380	1,740	300	961
Adblock Plus	59	100	20	20	0	100	900	3,300	0	0	0	0
Skype	82	74	140	130	100	300	900	3,300	130	250	141	277
Avira Browser Safety	94	90	60	50	100	200	1,100	3,600	0	110	63	112
Avast SafePrice	99	68	120	90	100	200	1,100	4,000	310	400	67	82
AdBlock	100	100	50	20	0	100	1,000	3,300	0	0	0	0
Google Translate	100	100	20	20	0	100	900	3,200	0	0	0	0
Pinterest Save Button	100	100	30	30	0	100	800	3,200	0	0	0	0
Tampermonkey	100	100	20	20	0	100	1,000	3,400	0	0	0	0
uBlock Origin	100	100	20	20	0	100	1,000	3,600	0	0	0	0
Spartacus	**100**	***100***	**20**	***20***	**0**	***100***	**800**	***3,200***	***0***	***0***	***0***	***0***

Table 5.6: Exthouse Metrics of Top 10 Chrome Extensions [1] along with Spartacus When Visiting Benign and Malicious Websites.



(a) Default browser visit. (b) Spartacus visit

Figure 5.7: Difference Due to the Shape of Buttons.



(a) Default browser visit. (b) Spartacus visit

Figure 5.8: Difference Due to Popup.

Chapter 6

ICORE: CONTINUOUS AND PROACTIVE EXTROSPECTION ON MULTI-CORE IOT DEVICES

Inspired by the proactive mechanism such as Spartacus that prevents users from seeing sophisticated phishing websites, I also think of a similar methodology to protect the OS kernels of IoT devices. By design, the secure world is a slave of the normal world even though it has the highest priority in an IoT device. The security functionalities in the secure world may not be invoked if the attacker that compromised the normal world chose not to. To overturn the master-slave role between the normal and secure world, I present iCORE to have the secure world independently and proactively monitor the operations and activities of the IoT device.

6.1 Assumptions and Threat Model

6.1.1 Assumptions

We assume that iCORE is implemented on IoT devices deploying multi-core ARM platform with ARM TrustZone extension. We consider the secure world as trusted and the normal world as our monitoring target. Also, we assume that the system will not be compromised or attacked during the cold boot procedure. To prevent the system from being tampered during the booting time, ARM secure boot [69] procedure can be leveraged to initialize the processors in order to guarantee the integrity of the whole memory area.

6.1.2 Threat Model

The normal world OS kernel mainly encounters the threats from kernel level rootkits. As aforementioned, the normal world OS is vulnerable and can be compromised any time

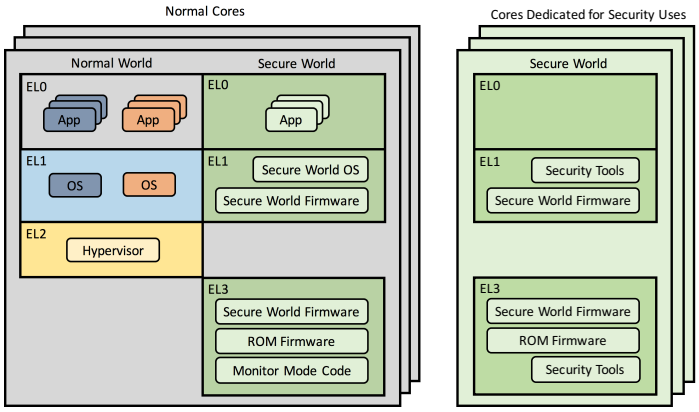


Figure 6.1: ICORE Architecture.

after the system has booted [171]. Therefore, kernel-level rootkits can be installed to take control of and attack the normal world OS. To hide the evidence of their intrusion, rootkits will tamper with some parts of the OS kernel such as setting a hook in certain system calls, which reside in the static kernel memory area.

Furthermore, attackers may notice that some security tools reside in the normal world OS. To avoid the detection from the security tools, the rootkits can deploy *transient attacks* [30], which do not tamper with the system permanently, in the kernel. By exploiting transient attacks, rootkits mitigate the permanent modifications on the kernel memory, which sets a camouflage to cheat on the security tools. However, iCORE checks the integrity of the static kernel memory area of the normal world continuously. Any modification on the monitored area will be detected. Consequentially, temporary modifications through transient attacks can still be detected by iCORE.

6.2 System Design

In this section, we present iCORE, a novel continuous and proactive extrospection with high visibility on IoT devices to monitor and detect the integrity of the static kernel memory in the normal world.

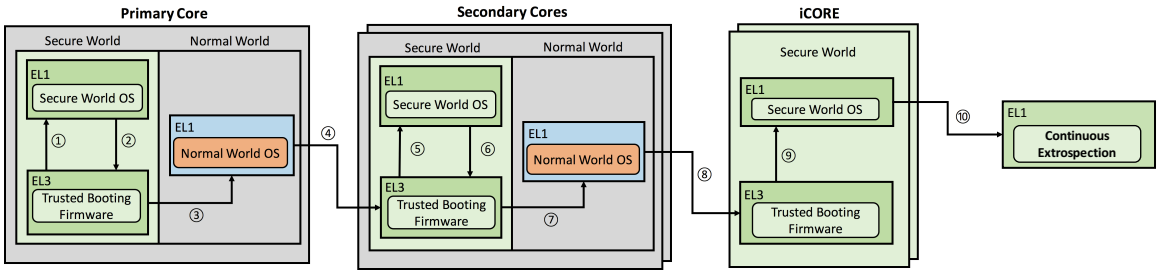


Figure 6.2: Core Initialization with iCORE.

Figure 6.1 illustrates the architecture of iCORE. Dedicated cores assigned as iCORE for security uses are deployed and running only in the secure world, meanwhile, the other cores are processing the conventional workload with the access of both normal and secure worlds. Security tools of iCORE are residing in exception level EL1 in the secure world OS. With the privilege of EL1, iCORE has the right to access the normal world kernel memory that has the same exception level as iCORE. According to our design, iCORE can access and monitor the static kernel memory of the normal world continuously and proactively without notifying or getting permission from the normal world.

In the following subsections, we demonstrate how dedicated cores are chosen as iCORE and initialized during a modified booting procedure deployed in the secure world. Meanwhile, cryptographic checks are applied to each stage of the secure world boot procedure, pointing to protect the integrity of the secure world image from unauthorized and malicious modification. Then, we discuss a mechanism that iCORE acquires data stored in the static kernel memory region of the normal world and checks the integrity of the specific memory area with the acquired data proactively and continuously.

*6.2.1 Initialization of i*CORE

Figure 6.2 shows the flow chart of iCORE's initialization. The sequence of core initialization is from the primary core to the secondary cores and finally to iCORE. When

the device is cold booted, the primary core is first invoked by a trusted booting firmware, for example, ARM-TF, and initialized in the secure world with exception level EL3. The primary core executes the boot path in the trusted booting firmware following such steps: Application Processor Trusted ROM (Boot Loader Stage 1, BL1), Trusted Boot Firmware (BL2), and EL3 Runtime Software (BL3). After the initialization of primary core is finished by the trusted firmware with EL3, the firmware then switches the control to the secure world OS with EL1 as shown in Step ① to continue the initialization of the primary core. In Step ②, the secure world OS should switch back to the trusted booting firmware after it finishes primary core initialization to perform the world switch operation with `smc` instruction through the trusted firmware. After the normal world OS gains the control from the secure world in Step ③, the primary core will have its initialization finalized in the normal world. The process of initializing secondary cores is similar to that of the primary core. After the primary core finishes its initialization in the normal world OS, a switch to the secure world that the trusted booting firmware generates to wake the secondary cores up, as shown in Step ④. Secondary cores are also initialized through the firmware to the secure world OS (Step ⑤), switching the world by the trusted booting firmware (Step ⑥), and eventually to the normal world OS (Step ⑦).

The regular cold boot procedure of an ARM multi-core platform indicates that the initialization of each core goes through the secure world and the normal world. As discussed in Section 6.1, the normal world can be compromised anytime so that the normal world is our monitoring target and cannot be trusted. To prevent iCORE from being infected, iCORE should never go to the normal world from the beginning of the initialization. To achieve this goal, we modify the boot procedures of iCORE by redirecting the core sequence to execute the monitoring functions. After the secondary cores finish the initialization in the normal world, a world switch to the secure world (Step ⑧) occurs to start the initialization of iCORE. iCORE is first initialized in the trusted booting firmware and then goes to the

secure world OS to continue its booting (Step ⑨), as what other cores do. However, iCORE never switches to the normal world to finalize the initialization. Instead, it dedicates itself to stay in the secure world forever by executing security functionalities (Step ⑩) to monitor the normal world kernel memory and to detect potential malicious modification.

The normal world OS kernel waits for the cores to come online or time out. For example, when the normal cores switch to the normal world eventually, the normal world OS detects the normal cores online within certain time limit and notifies the successful detection to the system. When the normal world OS tries to detect iCORE, however, iCORE never returns to the normal world. After the time exceeds the threshold, the normal world OS stops waiting the dedicated cores to execute the following operations but informs the system during the booting period. Therefore, it does not influence the normal initialization and execution of the system that iCORE never returns to the normal world. By now, iCORE officially gets out of control of the normal world to be able to execute the monitoring functionalities independently and proactively.

In summary, by initializing and staying only in the secure world, iCORE can avoid the detection of potential attacks coming from the normal world. The normal world can still continue its conventional executions without iCORE coming online.

6.2.2 Continuous and Proactive Extrospection

To provide continuous and proactive extrospection with high visibility on the static kernel memory blocks of the normal world, iCORE needs to access the designated memory area, load the data, and check the integrity. Figure 6.3 shows the process of the extrospection of iCORE.

Though as originally designed in ARM TrustZone the secure world has the privilege to access all the resources in the normal world, which facilities iCORE to protect the normal world with high visibility, two features form an obstacle for iCORE. The first feature is that

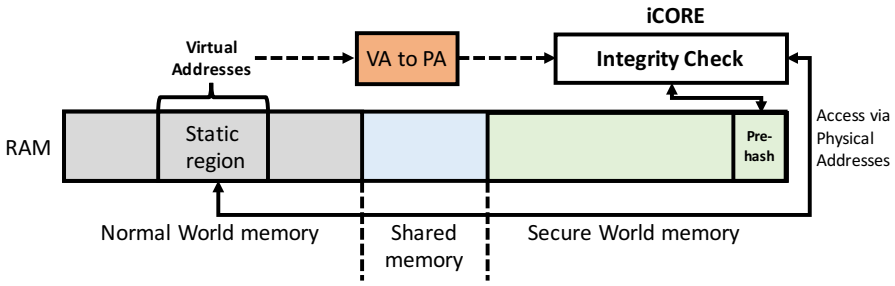


Figure 6.3: Extrospection of ICORE.

the normal world memory is not yet registered in the secure world, but the secure world can only access registered memory region. And the other one is that the addresses of kernel memory area are stored as virtual ones, the secure world, however, exploits the physical addresses to access the normal world memory. If iCORE plans to monitor one specific kernel memory region of the normal world, the corresponding memory region should be properly registered in the secure world beforehand and the starting and ending physical addresses must be obtained by converting the virtual addresses that the normal world use. However, ARM architecture does not offer any facilities for the secure world to register the normal world memory or to convert the virtual addresses. Hence, we design a function for iCORE by traversing a given virtual address on the normal world translation tables to retrieve the corresponding physical address. Additionally, we modify the configuration of memory layout to register the normal world memory in the secure world before the system boots. The detailed implementation of registering memory and converting virtual addresses will be discussed later in Section 6.3. Afterwards, iCORE can load the data used for integrity checking from the specific static kernel memory region of the normal world.

iCORE checks the integrity by the following steps. First, when the normal world OS is waiting for iCORE online during its initialization, iCORE loads the data from the monitored memory region and calculates a hash value of the data, named as `pre-hash`. Based on the assumption in Section 6.1, the attackers cannot compromise the normal world OS until the

system is completely booted. Therefore, we can trust `pre-hash` as the root of trust. Also, `pre-hash` is stored in the secure world memory that cannot be accessed by the normal world and any trusted applications because trusted applications are in EL0, lower privilege than that of iCORE, EL1. Next, when the normal world OS is ready and the user processes start to execute, iCORE again accesses the same memory region and retrieve the hash value, named `current-hash`. Then iCORE compares `current-hash` with `pre-hash`. If these two values are equal, it indicates that the integrity of the monitored memory region is guaranteed. Otherwise, iCORE will report it as tampered memory area with memory dumped to be further analyzed.

Such extrospection provided by iCORE is continuous. iCORE is repeatedly computing `current-hash` of the monitored area and comparing those two values to guarantee the integrity. Moreover, the monitoring functionalities of iCORE are proactive. iCORE is completely running independently without permissions or requests from the normal world since it has got rid of the control of the normal world during the initialization.

6.3 Implementation

We implemented the prototype of iCORE and tested it on a Hikey LeMaker board, which has 8 ARM Cortex-A53 1.2GHz cores and 2GB of memory. Regarding the software stack of our testing environment, the secure world side runs OP-TEE version 2.0.0 [172], combined with ARM-TF [68], while the normal world runs Linux distribution with kernel version 4.15. we open source the prototype with the expectation that it will be exploited and extended further by security researchers [1] .

[1] https://github.com/FormerBuckeye/iCore.git

Listing 6.1: Code of core initialization in iCORE.

```
LOCAL_FUNC vector_cpu_on_entry , :
    ...
    bl      get_core_pos

    /* begin to select iCORE*/
    cmp     x0, #7
    beq     iCORE_func
    /* end*/

    ...
    smc     #0
    b       .       /* SMC should not return */
    END_FUNC vector_cpu_on_entry
    ...
    LOCAL_FUNC iCORE_func , :

    /* execute functionalities of iCORE */
    b       do_extrosepction
END_FUNC iCORE_func
```

6.3.1 Core Initialization

As mentioned earlier in Section 6.2, all the cores are firstly initialized in the secure world by the ARM-TF and then the secure world OS, in our case, OP-TEE. Eventually, all the cores are returned to the normal world to finalize the initialization by the normal world OS. We analyze the source code of OP-TEE and modify it to select dedicated cores as iCORE. In OP-TEE source code, Function `vector_cpu_on_entry()` in File `thread_a64.S` is responsible for switching from the secure world to the normal world using `smc #0` instruction after the core initialization is finished in the secure world. The detailed iCORE selection procedure we implement has been shown in Listing 6.1.

Function `get_core_pos()` in Line 3 gets the number of the current core and stores the result to Register `x0`. Line 5-6 is the modified code to assign dedicated cores. In our implementation, we select one secondary core as iCORE from the perspective of performance efficiency. There are 8 cores in the Hikey LeMaker board, one primary core numbered as 0 and seven secondary cores numbered through 1 to 7. We choose the core No.7 as iCORE because as discussed in Section 6.2, iCORE will be booted after all normal cores finish initialization. When core No.7 is initializing, Line 6 forces it to execute

Function `iCORE_func()` to invoke iCORE functionalities, instead of executing `smc #0` to switch back to the normal world. In the normal world side, Function `_cpu_up()` in File `cpu.c` of the normal world OS is waiting for core No.7 to switch to the normal world. After time exceeds the limit, `_cpu_up()` will only notify the system that core No.7 fails to come online. Sequentially, the normal world OS ignores the offline core and continues to execute the following operations. By now, iCORE loses the control of the normal world and can perform the monitoring functionalities proactively and independently. The whole initialization does not require any changes on the normal world OS, which lighten the workload for developers to deploy iCORE architecture.

6.3.2 Memory Acquisition

After iCORE gets initialized and started to execute the functionalities, the static kernel memory region of the normal should be properly acquired by iCORE to load data. To make the secure world access the static kernel memory region of the normal world, as aforementioned in Section 6.2, two tasks should be resolved. First, the normal world memory should be correctly registered in the secure world. Second, the virtual addresses used by the normal world should be converted to the corresponding physical addresses for the secure world to use.

We conquer the first task by modifying the boot configuration memory map in the secure world OS. Specifically, the secure world exploits `bootcfg_memory_map` to manage the memory layout provided to the TEE core. `bootcfg_memory_map` is a structure type of `map_area` to record the memory area registered in the secure world. In this structure, every memory area that the secure world needs to access is listed along with its type (all types of memory area have been enumerated in `mmu.h`), starting physical address, size, and attribute. Hence, we register the normal world memory to the secure world by adding the corresponding memory layout in `bootcfg_memory_map`. List-

Listing 6.2: Memory registration.

```
static struct map_area bootcfg_memory_map[] =
{
        {
                .type = MEM_AREA_NSEC_SHM,
                .pa = DRAM0_BASE, .size = DRAM0_SIZE,
                .cached = true, .secure = false,
                .rw = true, .exec = false,
        },
        ...
}
```

ing 6.2 demonstrates the memory layout that we insert. The type of the memory layout is `MEM_AREA_NSEC_SHM`, which represents that the normal world memory now is registered as non-secure shared RAM between the normal world and the secure world; the starting physical address is `DRAM0_BASE`, which is the base address of the DRAM of the normal world; the size is `DRAM0_SIZE`, which is the size of DRAM of the normal world; and it can be cached, can be read and written, cannot be executed, and is non-secure.

Afterwards, we should solve the second task. There is no related instruction or function for the secure world to convert the virtual address to the physical address by default. In the normal world, one process accesses the memory by providing the virtual address in its own private space to MMU, and MMU then looks up the page tables to calculate the physical address. The secure world has to traverse the same page tables to convert the virtual address. Hence, we implement the converting function called `va2pa_in_sec()` in iCORE before accessing the memory of the normal world. Specifically, for a given virtual address in AArch64, the most significant bits determine the base address of the page table, and then bits [41:29] and [28:16] are the index of Level 2 and Level 3 page tables to find the corresponding page table entry level by level. Finally, we combine the bits [47:29] of the page table entry and the least significant bits of the virtual address to get the proper physical address.

6.3.3 Continuous and Proactive Extrospection

Finishing two tasks mentioned above, iCORE has the ability to monitor the normal world kernel and to check its integrity. As the static data in the normal world kernel memory is consecutively stored, iCORE can load data in the static memory regions of the normal world kernel by accessing from the starting to the ending physical addresses. However, iCORE can only obtain the virtual addresses of the corresponding memory area by analyzing File `system.map`, which is generated during the kernel compiling and records the virtual addresses of symbols used in the normal world kernel. Function `va2pa_in_sec()` is then exploited to convert the virtual addresses to physical addresses to help iCORE access the data.

To measure the integrity of static data and code, first iCORE deploys SHA-1 cryptographic hash algorithm to compute the root of trust, `pre-hash`, of the loaded data before the normal world OS finishes the initialization. The `pre-hash` is then stored in the secure world memory. Second, when all the processes starts to execute, iCORE loads the data from the same normal world kernel memory region and computes the `current-hash`. Next, iCORE compares `current-hash` with `pre-hash`. The result of two hash-value comparison will determine whether the static kernel memory of the normal world has been tampered with.

To provide continuous monitoring functionalities, except the first step (because `pre-hash` is the root of trust), others should be processed repeatedly. iCORE leverages an infinite loop to achieve the continuous extrospection. Thus, in each loop iCORE can check the integrity by calculating and comparing the hash values. All the monitoring procedures are also executed by iCORE independently, without requesting or getting permission from the normal world, because the normal world cannot detect the existence of iCORE. Consequently, iCORE shows continuousness and proactiveness to detect the malicious modifi-

cation in the monitored memory areas, and will not be affected or disabled by the normal world.

6.4 Evaluation

After implementing iCORE, we answer the following questions that may be asked: How effective is iCORE? Can iCORE be detected or even terminated from the normal world? Losing one core brings a negative impact on the performance of IoT and mobile devices, can such a device be qualified to meet the daily requirements?

*6.4.1 Effectiveness of i*CORE

To evaluate the effectiveness of iCORE to protect the static kernel memory area of the normal world against the attacks and vulnerabilities that are probably exploited, we design experiments trying to (1) tamper the monitored memory area, and (2) terminate iCORE from the normal world. To experiment the first task, we deploy Loadable Kernel Module in the normal world attempting to read and write the monitored static kernel memory area with root privilege. Unsurprisingly, iCORE detects all the modifications on the memory region. It indicates that iCORE can detect any malicious modification on the static kernel memory area of the normal world from potential attacks such as malicious code injection [173] and vulnerabilities that are exploited to tamper with the static kernel memory such as allowing privileged users to modify a limited range of kernel memory in syscall interface of bridging [174].

Secondly, to verify that iCORE cannot be detected or disabled from the normal world, we check the `cpuinfo` of the normal world OS that records the CPU information. The content in `cpuinfo` only shows the information of CPU 0 to 6. It indicates that the normal world cannot detect the existence of iCORE and thus it does not provide any methods for attackers in the normal world to detect or even disable iCORE.

Table 6.1: Performance Overhead When Checking Different Size of Static Memory Area.

Benchmark application	Monitoring whole static memory	Monitoring specific static memory
perlbench	2.28%	1.25%
mcf	3.61%	2.42%
omnetpp	3.14%	2.02%
xalancbmk	12.50%	2.79%
x264	2.38%	0.77%
deepsjeng	1.17%	1.01%
leela	1.92%	0.73%
xz	3.95%	2.64%

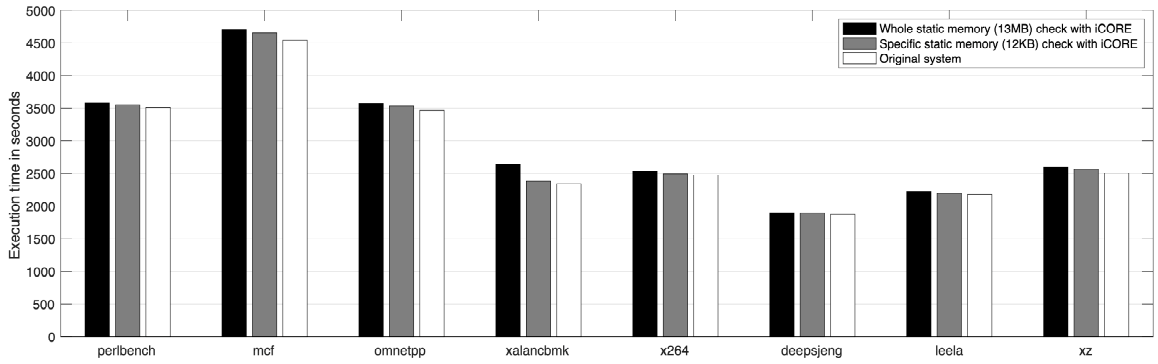


Figure 6.4: Evaluation Result in SPEC CPU2017.

6.4.2 Performance with iCORE

iCORE may impact the performance of the device by frequently checking the whole static kernel memory of the normal world. Besides the impact of losing one core, frequent memory access may influence the speed of workload execution as well. Reading normal world memory may fill up L2 cache shared among cores and the data bus between CPUs and memory. Therefore, the data required by processes in the normal world would be delivered slower than the original system. We evaluate the performance of the system with iCORE by checking the whole static kernel memory of the normal world and by checking

a portion of kernel text (code) area to determine how the impact of iCORE scales when it monitors different sizes of static kernel memory. According to our analysis, the size of whole static kernel memory is around 13MB while that of selected text (code) area is about 12KB.

We evaluate the performance of the normal world OS using the `SPEC CPU2017` to measure the impact of permanently losing one core to the secure world. Figure 6.4 represents the evaluation result under 3 situations, a system with 7 cores and iCORE checking the whole static memory of the normal world, one with 7 cores and iCORE checking a small portion of the static memory, and the original system with 8 cores, in `SPEC CPU2017`'s 8 benchmarks of `intrate` suit. Table 6.1 shows the performance overhead of the system when checking a different size of the static kernel memory area of the normal world. From the evaluation result, iCORE unsurprisingly brings the overhead on CPU performance of the system because of losing one core and frequently accessing the memory. When checking the whole static kernel memory of the normal world, iCORE generates 1.17% (*deepsjeng*) to 12.50% (*xalancbmk*) performance overhead compared with the original system.

According to the analysis of `SPEC CPU2017` in [175], benchmark *xalancbmk* has the highest percentage of branch instructions, over 30%, while the benchmarks *omnetpp*, *leela*, and *deepsjeng* only have 15% of branch instructions. Besides, *xalancbmk* consumes a considerable percentage of their execution time on cache and memory related operations. This explains why iCORE has a more overhead fraction on operating *xalancbmk*: executing this benchmark application requires much more memory/cache accesses than operating the other ones; however, the system with iCORE does not provide as much computation power as the original system does because it only has 7 cores to process the workload in the benchmark, and iCORE uses L2 cache lines and data bus between CPU and memory for the monitoring purpose.

We also evaluate that monitoring smaller portion of static kernel memory of the normal

world, which brings the system less performance degradation as shown in Figure 6.4. The performance overhead of checking 12KB static memory on *xalancbmk* is 2.79% compared with that on the original system. The CPU performance increases, but not significantly, even though iCORE only checks 0.1% of the whole static kernel memory of the normal world. One possible reason is that the Linux scheduler helps the system work efficiently by checking the load balancing of each CPU dynamically and distributing the routine workload to the low CPU usage core so that the tasks can be executed as fast as possible.

As a consequence, the system with iCORE, checking even the whole static kernel memory, can still finish the same workload within the negligible performance overhead.

6.5 Discussion and Future work

6.5.1 Response After Detection

After detecting the malicious modification in the monitored memory region of the normal world, iCORE should have corresponding measures to report and analyze the memory area. However, there exists a semantic gap that iCORE only detects the modification but has no ability to extract the semantic information such as the current CPU info, the active processes, or potential hooks on system calls. iCORE has to get a hand from memory forensic tools [176] developed to analyze and extract digital artifacts.

Memory forensic tools generally require the memory sample file, which iCORE can provide. If there are malicious modifications occurred under the monitoring of iCORE, the normal world memory can be dumped since iCORE has the ability to access the whole normal world resources. Afterwards, the dumped memory file will be sent to memory forensic tools outside the device. We can leverage one of the memory forensic tools such as Volatility [177]. The memory forensic tools, then, can analyze the memory sample and extract semantic-rich information, such as the active process list, potential hooks on system

Table 6.2: Comparison of the Related Works with iCORE.

Criteria	kGuard [53]	SIM [55]	NOVA [178]	Vigilare [30]	MIPE [179]	TZ-RKP [20]	SPROBES [21]	SecVisor [65]	iCORE
Resides out of the monitored system	○	●	●	●	●	●	●	●	●
Provides proactive monitoring	●	●	●	○	○	○	○	●	●
No need for virtualization supports	●	○	○	●	●	●	●	○	●
No need additional hardware	●	●	●	○	●	●	●	●	●
No need to modify monitored system	●	●	●	●	○	○	○	●	●
In-time tamper detection	○	○	○	○	○	○	○	○	●

●: satisfying the criteria, ○: not satisfying the criteria.

calls, and a list of opened files by the normal world OS kernel. Therefore, by means of the memory forensic tools, the semantic gap between iCORE and the normal world can be narrowed down. Also, the further analysis after detecting malicious modifications can be possible.

*6.5.2 Potential Threats Against i*CORE

iCORE is designed to detect malicious modifications on the static kernel memory area of the normal world. However several types of threats can be exploited to potentially hazard iCORE.

First, during the booting procedure of a system with iCORE, a threat that the attacker is attempting to modify the kernel image file may occur. To protect the kernel binary from maliciously tampering, we implement secure boot procedure [69] on ARM platform that is mentioned in Section 6.2. Hence, secure boot procedure guarantees that iCORE can be booted safely. Once iCORE is booted, the attacker cannot detect or turn off the functionalities of iCORE, because it is completely isolated from the monitored system.

Secondly, as we mention in Section 6.2, to load memory data, iCORE exploits `system.map` to get the starting and ending address of the static kernel memory area of the normal world. Some may concern that the addresses where the static kernel data is stored can be modified by the attackers so that iCORE cannot work properly with incorrect addresses. This is avoidable because File `system.map` is generated before the system

booting. Static kernel data of the normal world will be loaded according to the file during the initialization of the system. Before the attacker takes control of the normal world, iCORE has already gained the physical addresses of the monitored memory area. Hence, the changes of the addresses do not influence the correctness of iCORE's functionalities.

Thirdly, when iCORE detects the malicious modification on the monitored memory area, the normal world memory can be dumped for further analysis. However, the normal world RAM could be tampered again by the attacker to erase the evidence of crimes. To avoid the situation mentioned above, TrustDump [180] provides an approach to allow the device user to switch into the secure world safely when the normal world OS crashes or is compromised in order to cease the operations in the normal world. Therefore, the normal world memory cannot be further tampered with and the memory dumped by iCORE can be trusted.

*6.5.3 Extension of i*CORE

Our current design of iCORE checks the static code and data in the normal world kernel memory. However, the attackers would attempt to tamper with the dynamic code to make the attacks successful. Unlike the static code and data, it is much more difficult to check the integrity of the dynamic data memory region, as the data is changing all the time when the system is running. It is a challenge for an integrity check to distinguish between a normal operation modification and a potential tampering behavior. Petroni et. al [181] propose an architecture to provide the integrity to dynamic kernel data using a specification language-based approach. Also, the code-reuse attacks are popular now which execute the existing code in the memory instead of code injection to by-pass the integrity check of security tools. In the future, we will develop iCORE to monitor the dynamic memory area and protect the normal world from the code-reuse attacks.

In addition to the integrity check improvement, iCORE will be able to overcome the

semantic gap. As aforementioned, iCORE can exploit the data read from the static memory region in the normal world to check the integrity. However, the real meaning of the data iCORE reads remains unknown, so we can leverage the existing memory forensic tools to extract the semantic-rich info from the memory dump file. Nevertheless, analyzing the dump file in device is necessary because transferring the dumped memory file out of the device can be disabled by the attacker if she has the physical control of the device. The following work of iCORE will be attempting to narrow the semantic gap itself to enforce the security since the bridge between binaries and the kernel structures will help iCORE to monitor the real critical information.

6.6 Related Work

Hardware/Hypervisor-assisted Monitoring Methods. As we mentioned in Chapter 2, modern monitoring methods are classified into hardware-assisted and hypervisor-assisted methods, the shortcomings and advantages of both of which have also been discussed. Other efforts have been made to expand the monitoring fields and to mitigate the negative impact brought by these two categories.

Articles [182, 183, 178] proposed another method called tiny-hypervisor. It is a thin software TCB (trusted computing base) that has a small amount of code to reduce the attack interface to monitor and protect the system, which performs well on monitoring virtual machines. However, they still require virtualization support on the target system, which is costly deployed on all mobile devices. Since most mobile and IoT devices are deploying ARM TrustZone extension, iCORE will have a better performance than the tiny-hypervisors do.

As for hardware-assisted methods that we mentioned in Section **??**, Vigilare [30] and Ki-Mon [31] provided additional external equipment to protect the static and dynamic kernel objects from being tamped, respectively. Davi et. al [184] designed a hardware-based

control flow integrity defense for embedded systems against return-oriented programming (ROP) attacks. Based on this work, HAFIX [185] is proposed as a defensive extension against ROP attacks using backward edges. However, it is costly and inconvenient to take another device dedicated to real-time protection and monitoring purposes along with the mobile devices. To avoid using additional equipment, TZ-RKP [20] and SPROBES [21] take advantage of ARM TrustZone extension to trap the page table updates, switch to the secure world through `smc` instruction, and verify the write signal on the monitored memory area. MIPE [179] does the similar work to protect the memory of the normal world. It checks in the secure world if the previous status of a physical address that the normal world maps when a page fault occurs has already been mapped to a virtual address. These methods still require operations in the normal world to switch to the secure world, which slows the execution down and can be disabled by the compromised normal world. iCORE provides a possibility to the research on developing security tools based on ARM TrustZone that the monitoring tool in the secure world can struggle out of control of the normal world with self-decision-making power so that the attackers in the normal world cannot disable its functionalities.

Continuous Monitoring Mechanisms. Current continuous monitoring mechanisms exploit trampolines, which are used to store addresses pointing to interrupt service routines, or hooks pre-installed in the monitored system to change the control flow of the existing execution path to the monitoring functionalities. This requires that the hooks and trampolines should be properly protected from tampered so that the monitoring mechanism can keep their integrity and availability. Articles [58, 55] implement trampolines that are guaranteed by a hypervisor level memory protection approach to monitor the VM.

From another perspective, ARM TrustZone is also exploited to deploy the continuous monitoring. Mechanisms such as TIMA (TrustZone Integrity Measurement Architecture) [186, 187] implement the monitoring code in TrustZone and implant the trampoline

in the secure world kernel to enhance the integrity of the static memory block in the normal world. The purpose of iCORE is also to provide a secure environment based on ARM TrustZone for normal world OS. However, iCORE is more effective because it does not require additional modifications on the normal world OS source code and can get access to all the normal world memory with high visibility due to its highest privilege state. Thus, iCORE is an improvement of the traditional continuous monitoring mechanism.

6.7 Conclusion

In conclusion, we present iCORE, an innovative continuous and proactive extrospection system with high visibility technique on IoT devices deploying multi-core ARM platform in this paper. iCORE exploits ARM TrustZone technology to dedicate one core in the secure world forever, assuring the computing integrity of static kernel memory region of the normal world. By breaking the original time-sharing paradigm of such systems, iCORE enables continuous co-processor-like monitoring with high visibility into the rich execution environment on such mobile and IoT platforms based on the design of ARM TrustZone architecture that the secure world can access all the resources in the normal world. iCORE plays a role as master and monitors the pre-selected memory area proactively so that the normal world cannot detect or disable the functionalities of it because iCORE gets out of control of the normal world since the system booting procedure. And by ensuring that security tools execute on certain physical CPU cores, the system's attack surface is significantly reduced. Also, with the increasing number of mobile CPU cores and based on the results of the evaluation, iCORE only introduces a negligible overhead.

Chapter 7

CONCLUSION

7.1 Attacks Can Be mitigated

Through the work presented in this dissertation, I have analyzed cyber attacks from different types — phishing and IoT attacks. The analysis results offer critical inspirations to the security community and hence the community can propose mitigations accordingly. The solutions based on my analysis of cyber attacks have clearly showed paths to improve the defenses against emerging and advanced attacks, such as anti-phishing blacklists, machine-learning based anti-phishing systems, back-end anti-cloaking mechanisms, in-browser defense, and monitoring in IoT devices.

I started my analysis from the nature and prevalence of server-side and client-side cloaking techniques in phishing attacks, which provided me an insight that the current anti-phishing ecosystem has limited solutions mitigating such advanced attack. So I designed, implemented, and evaluated CrawlPhish to force the client-side cloaked phishing websites to reveal the hidden malicious content. To prevent phishing attacks, I leveraged the fingerprinting feature in server-side cloaking techniques and proposed Spartacus. It deliberately triggers server-side cloaking techniques in phishing websites and hence users will not see the phishing content. Therefore, the ecosystem has state-of-the-art methodologies to fight against server-side and client-side cloaking techniques.

Next, in PHISHPATH, I analyzed another technique that criminals implement in phishing attacks — redirection, to evade automated detection. Phishers use redirection techniques differently from those in benign websites. This finding inspires me to leverage such difference to identify phishing redirection chains out of benign ones. Hence, I developed a

machine learning based classifier, which only uses redirection features, and it can precisely distinguish malicious and benign redirection chains. I also provided a recommendation to blacklist based anti-phishing systems that they need to block intermediate domains and hence the landing hops that use those intermediate ones will be inaccessible automatically. The mitigations I proposed to the ecosystem greatly increase the speed and accuracy of identifying and blocking phishing redirection chains in a large scale.

Finally, inspired by Spartacus, the IoT devices can proactively extrospect activities in the normal execution environment. So I designed iCORE to overturn the master-slave relationship between the normal and secure world. Hence, with the highest privilege in the platform, the secure world can independently monitor all the operations in the device even though the device is compromised.

7.2 Remaining Challenges

Even though the modern security society represents a significant improvement over cyber attacks such as phishing and IoT compromise, criminals can leverage the similar ideas under different context.

For example, there are advertisements in the social media who leverage cloaking techniques to fool detection systems. They display shopping web pages with legal items to the ad host, but sell users who click in prohibited goods such as drug and weapon. Furthermore, as the concept of metaverse emerges, similar attacks can be deployed to tackle users in the metaverse society. Phishing attacks can steal the credentials from users and transfer their NFTs (Non-Fungible Token), which may generate more loss than the current stage of phishing. The attacks focusing on compromising systems can also be deployed on the new devices such as VR (Virtual Reality) devices because users need them to experience the metaverse.

I hope that beyond improving defense methodologies in the current platforms such

as IoT devices and emails/text/social media posts, the security society should expand the methodologies to other architectures or concepts. By closing the gap between existing and new platforms, criminals cannot simply deploy an existing type of attacks on the new platforms. If the attacks cease being sufficiently profitable, criminals will likely focus on other type of attacks.

REFERENCES

[1] Treo. Exthouse: Analyze the impact of a browser extension on web performance. , 2020. `https://github.com/treosh/exthouse`.

[2] Google. Google transparency report. 2019. https://transparencyreport.google.com/safe-browsing/overview?hl=en.

[3] Anti-Phishing Working Group: APWG Trends Report Q1 2018, 2018. (Date last accessed 31-August-2018).

[4] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Gary Warner. Inside a phisher's mind: Understanding the anti-phishing ecosystem through phishing kit analysis. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–12. IEEE, 2018.

[5] Adam Oest, Yeganeh Safaei, Adam Doupé, Gail-Joon Ahn, Brad Wardman, and Kevin Tyers. Phishfarm: A scalable framework for measuring the effectiveness of evasion techniques against browser phishing blacklists. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (Oakland)*, pages 764–781, Oakland, CA, May 2019.

[6] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, et al. Data breaches, phishing, or malware?: Understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1421–1434. ACM, 2017.

[7] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean-Michel Picod, and Elie Bursztein. Cloak of visibility: Detecting when machines browse a different web. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 743–758. IEEE, 2016.

[8] Nektarios Leontiadis, Tyler Moore, and Nicolas Christin. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. In *USENIX Security Symposium*, volume 11, 2011.

[9] Davide Canali, Davide Balzarotti, and Aurélien Francillon. The role of web hosting providers in detecting compromised websites. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 177–188, New York, NY, USA, 2013. ACM.

[10] Brett Stone-Gross, Thorsten Holz, Gianluca Stringhini, and Giovanni Vigna. The underground economy of spam: A botmaster's perspective of coordinating large-scale spam campaigns. *LEET*, 11:4–4, 2011.

[11] Aditya K Sood and Richard J Enbody. Crimeware-as-a-service: a survey of commoditized crimeware in the underground market. *International Journal of Critical Infrastructure Protection*, 6(1):28–38, 2013.

[12] Kevin Borgolte, Christopher Kruegel, and Giovanni Vigna. Meerkat: Detecting website defacements through image-based object recognition. In *24th USENIX Security Symposium USENIX Security 15)*, pages 595–610, 2015.

[13] Brad Wardman, Dan Clemens, Joseph Wolnski, PayPal Inc-San Jose, Shadow Dragon-United States, and Shadow Dragon-United States. Phorecasting phishing attacks: A new approach for predicting the appearance of phishing websites. *Cyber-Security and Digital Forensics*, page 142, 2016.

[14] Kyle Soska and Nicolas Christin. Automatically detecting vulnerable websites before they turn malicious. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*, pages 625–640, 2014.

[15] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 128–138, 2007.

[16] Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho, and Sarah Martin. Trustzone explained: Architectural features and use cases. In *Proceedings of the IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 445–451, Pittsburgh, PA, 2016.

[17] AMD. Secure virtual machine architecture reference manual, 2005. `https://www.mimuw.edu.pl/~vincent/lecture6/sources/amd-pacifica-specification.pdf`.

[18] Intel. Intel trusted execution technology (intel txt), 2014. `https://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html`.

[19] Arijit Ukil, Jaydip Sen, and Sripad Koilakonda. Embedded security for internet of things. In *Proceedings of the 2nd National Conference on Emerging Trends and Applications in Computer Science*, Shillong, India, 2011.

[20] Ahmed M Azab, Peng Ning, Jitesh Shah, Quan Chen, Rohan Bhutkar, Guruprasad Ganesh, Jia Ma, and Wenbo Shen. Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, pages 90–102, Scottsdale, AZ, November 2014.

[21] Xinyang Ge, Hayawardh Vijayakumar, and Trent Jaeger. SPROBES: Enforcing kernel code integrity on the trustzone architecture. In *Proceedings of the 3rd IEEE Mobile Security Technologies Workshop (MoST)*, San Jose, CA, May 2014.

[22] Kyungtae Kim, I Luk Kim, Chung Hwan Kim, Yonghwi Kwon, Yunhui Zheng, Xiangyu Zhang, and Dongyan Xu. J-force: Forced execution on javascript. In *Proceedings of the 26th international conference on World Wide Web*, pages 897–906. International World Wide Web Conferences Steering Committee, 2017.

[23] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *Presented as part of the 22nd USENIX Security Symposium*, pages 637–652, 2013.

[24] Clemens Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: De-cloaking internet malware. In *2012 IEEE Symposium on Security and Privacy*, pages 443–457. IEEE, 2012.

[25] Charlie Curtsinger, Benjamin Livshits, Benjamin G Zorn, and Christian Seifert. Zozzle: Fast and precise in-browser javascript malware detection. In *USENIX Security Symposium*, pages 33–48. San Francisco, 2011.

[26] Marco Cova, Christopher Kruegel, and Giovanni Vigna. There is no free phish: An analysis of "free" and live phishing kits. *WOOT*, 8:1–8, 2008.

[27] Cynthia Kuo, Fritz Schneider, Collin Jackson, D Mountain, and T Winograd. Google safe browsing. project at google. *Inc., June–August*, 2005.

[28] Url and website scanner - urlscan.io. `https://urlscan.io/`.

[29] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)*, 14(2):21, 2011.

[30] Hyungon Moon, Hojoon Lee, Jihoon Lee, Kihwan Kim, Yunheung Paek, and Brent Byunghoon Kang. Vigilare: toward snoop-based kernel integrity monitor. In *Proceedings of the 19th ACM Conference on Computer and Communications Security (CCS)*, pages 28–37, Raleigh, NC, 2012.

[31] Hojoon Lee, Hyungon Moon, Ingoo Heo, Daehee Jang, Jinsoo Jang, Kihwan Kim, Yunheung Paek, and Brent Kang. Ki-mon arm: A hardware-assisted event-triggered monitoring platform for mutable kernel object. *IEEE Transactions on Dependable and Secure Computing*, 2017.

[32] Sun Bin, Wen Qiaoyan, and Liang Xiaoying. A dns based anti-phishing approach. In *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, volume 2, pages 262–265. IEEE, 2010.

[33] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. Lexical feature based phishing url detection using online learning. In *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, pages 54–60. ACM, 2010.

[34] Huajun Huang, Liang Qian, and Yaojun Wang. A svm-based technique to detect phishing urls. *Information Technology Journal*, 11(7):921–925, 2012.

[35] Mahmoud Khonji, Andrew Jones, and Youssef Iraqi. A novel phishing classification based on url features. In *2011 IEEE GCC conference and exhibition (GCC)*, pages 221–224. IEEE, 2011.

[36] Min Wu, Robert C Miller, and Greg Little. Web wallet: preventing phishing attacks by revealing user intentions. In *Proceedings of the second symposium on Usable privacy and security*, pages 102–113. ACM, 2006.

[37] Yue Zhang, Jason I Hong, and Lorrie F Cranor. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on World Wide Web*, pages 639–648. ACM, 2007.

[38] Haijun Zhang, Gang Liu, Tommy WS Chow, and Wenyin Liu. Textual and visual content-based anti-phishing: a bayesian approach. *IEEE Transactions on Neural Networks*, 22(10):1532–1546, 2011.

[39] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Ndss*, pages 1–17, 2011.

[40] Davide Canali, Davide Balzarotti, and Aurélien Francillon. The role of web hosting providers in detecting compromised websites. In *Proceedings of the 22nd international conference on World Wide Web*, pages 177–188. ACM, 2013.

[41] Colin Whittaker, Brian Ryner, and Marria Nazif. Large-scale automatic classification of phishing pages. In *Proceedings of the 28th Network and Distributed System Security Symposium (NDSS)*, 2010.

[42] Windows defender smartscreen. 2019. https://github.com/MicrosoftDocs/windows-itpro-docs/blob/public/windows/security/threat-protection/windows-defender-smartscreen/windows-defender-smartscreen-overview.md.

[43] Bin Liang, Miaoqiang Su, Wei You, Wenchang Shi, and Gang Yang. Cracking classifiers for evasion: a case study on the google's phishing pages filter. In *Proceedings of the 25th International Conference on World Wide Web*, pages 345–356, 2016.

[44] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. PhishTime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *Proceedings of the 29th USENIX Security Symposium*, 2020.

[45] David Y Wang, Stefan Savage, and Geoffrey M Voelker. Cloak and dagger: dynamics of web search cloaking. In *Proceedings of the 18th ACM Conference on Computer and Communications Security (CCS)*, pages 477–490. ACM, 2011.

[46] Ziji Guo. World-wide cloaking phishing websites detection. 2017.

[47] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, pages 281–290, 2010.

[48] Timofey Kachalov and zamotkin. Javascript obfuscator, 2016. https://github.com/javascript-obfuscator/javascript-obfuscator.

[49] APWG. Phishing Activity Trends Report 3rd Quarter 2019. 2019. `https://docs.apwg.org/reports/apwg_trends_report_q3_2019.pdf`.

[50] Anupama Aggarwal, Ashwin Rajadesingan, and Ponnurangam Kumaraguru. Phishari: Automatic realtime phishing detection on twitter. In *2012 eCrime Researchers Summit*, pages 1–12. IEEE, 2012.

[51] Mark Stockley. How scammers abuse google search's open redirect feature. 2020. https://nakedsecurity.sophos.com/2020/05/15/how-scammers-abuse-google-searchs-open-redirect-feature/.

[52] Heeyoung Kwon, Mirza Basim Baig, and Leman Akoglu. A domain-agnostic approach to spam-url detection via redirects. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 220–232. Springer, 2017.

[53] Vasileios P Kemerlis, Georgios Portokalidis, and Angelos D Keromytis. kguard: Lightweight kernel protection against return-to-user attacks. In *21st USENIX Security Symposium (USENIX Security 12)*, pages 459–474, 2012.

[54] Lionel Litty, H Andrés Lagar-Cavilla, and David Lie. Hypervisor support for identifying covertly executing binaries. In *Proceedings of the 17th USENIX Security Symposium (Security)*, pages 243–258, Boston, MA, 2008.

[55] Monirul I Sharif, Wenke Lee, Weidong Cui, and Andrea Lanzi. Secure in-vm monitoring using hardware virtualization. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 477–487, Chicago, IL, 2009.

[56] Xuxian Jiang and Xinyuan Wang. Out-of-the-box monitoring of vm-based high-interaction honeypots. In *Proceedings of the 10th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 198–218, Queensland, Australia, 2007.

[57] Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, 2005.

[58] Bryan D Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An architecture for secure active monitoring using virtualization. In *Proceedings of the 29th IEEE Symposium on Security and Privacy (Oakland)*, pages 233–247, Oakland, CA, 2008.

[59] MITRE. CVE-2018-7542 Detail, 2018. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-7542`.

[60] MITRE. CVE-2017-7228 Detail, 2017. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-7228`.

[61] MITRE. CVE-2017-15589 Detail, 2017. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-15589`.

[62] GlobalPlatform. GlobalPlatform made simple guide: Trusted Execution Environment (TEE) Guide, 2016. http://www.globalplatform.org/mediaguidetee.asp.

[63] Teresa F Lunt and R Jagannathan. A prototype real-time intrusion-detection expert system. In *Proceedings of the 9th IEEE Symposium on Security and Privacy (Oakland)*, pages 59–66, Oakland, CA, 1988.

[64] Zhi Wang, Xuxian Jiang, Weidong Cui, and Peng Ning. Countering kernel rootkits with lightweight hook protection. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 545–554, Chicago, IL, 2009.

[65] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)*, pages 335–350, Stevenson, WA, 2007.

[66] ARM. SMC CALLING CONVENTION System Software on ARM Platforms, 2016. `http://infocenter.arm.com/help/topic/com.arm.doc.den0028b/ARM_DEN0028B_SMC_Calling_Convention.pdf`.

[67] USMAN. Apple's Secure Enclave for Touch ID And Its Importance Detailed. 2013. `http://www.iphoneincanada.ca/iphone-5s/apples-new-secure-enclave-details/`.

[68] ARM. ARM Trusted Firmware, 2017. `https://github.com/ARM-software/arm-trusted-firmware`.

[69] ARM. ARM Security Technology Building a Secure System using TrustZone Technology, 2009. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.prd29-genc-009492c/index.html`.

[70] P Daniel, Cesati Marco, et al. Understanding the linux kernel, 2007.

[71] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *Proceedings of the 29th USENIX Security Symposium*, 2020.

[72] Eihal Alowaisheq, Peng Wang, Sumayah Alrwais, Xiaojing Liao, XiaoFeng Wang, Tasneem Alowaisheq, Xianghang Mi, Siyuan Tang, and Baojun Liu. Cracking the wall of confinement: Understanding and analyzing malicious domain take-downs. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2019.

[73] Xia-mu Niu and Yu-hua Jiao. An overview of perceptual hashing. *Acta Electronica Sinica*, 36(7):1405–1411, 2008.

[74] Richard W Hamming. Error detecting and error correcting codes. *The Bell system technical journal*, 29(2):147–160, 1950.

[75] MDN Web Docs. Mozilla web apis. 2020. https://developer.mozilla.org/en-US/docs/Web/API.

[76] Mozilla. Event reference, 2020. https://developer.mozilla.org/en-US/docs/Web/Events.

[77] Daniel St. Jules. Jsinspect: Detect copy-pasted and structurally similar code, 2017. https://github.com/danielstjules/jsinspect.

[78] Antawan Holmes and Marc Kellogg. Automating functional tests using selenium. In *AGILE 2006 (AGILE'06)*, pages 6–pp. IEEE, 2006.

[79] Katalon. Katalon studio, 2015. https://www.katalon.com/katalon-studio/.

[80] W3C. Web notifications. 2015. https://www.w3.org/TR/notifications/.

[81] Brian Anderson. Best automation testing tools for 2018 (top 10 reviews), 2017. https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2.

[82] Tim Rotolo. Mouse movement patterns and user frustration. 2016. https://www.trymyui.com/blog/2016/10/28/mouse-movement-patterns-and-user-frustration/.

[83] Yi-Min Wang and Ming Ma. Detecting stealth web pages that use click-through cloaking. In *Microsoft Research Technical Report, MSR-TR*, 2006.

[84] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[85] Rachna Dhamija, J Doug Tygar, and Marti Hearst. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 581–590. ACM, 2006.

[86] Ismail Cem Paya and Trevin Chow. Combining a browser cache and cookies to improve the security of token-based authentication protocols, July 3 2007. US Patent 7,240,192.

[87] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Rfc2616: Hypertext transfer protocol–http/1.1, 1999.

[88] Zhibo Sun, Carlos E. Rubio-Medrano, Ziming Zhao, Tiffany Bao, Adam Doupé, and Gail-Joon Ahn. Understanding and predicting private interactions in underground forums. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 2019.

[89] Ajay Modi, Zhibo Sun, Anupam Panwar, Tejas Khairnar, Ziming Zhao, Adam Doupé, Gail-Joon Ahn, and Paul Black. Towards automated threat intelligence fusion. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 408–416. IEEE, 2016.

[90] Claudio Guarnieri. The Year of the Phish. 2019. https://nex.sx/blog/212/15/the-year-of-the-phish.html.

[91] OpenPhish, 2014. https://openphish.com.

[92] PhishTank, 2006. https://phishtank.com.

[93] PhishStats, 2020. https://phishstats.info/.

[94] Google. Manual actions report. 2020. `https://support.google.com/webmasters/answer/9044175?hl=en&ref_topic=4596795`.

[95] Amazon. Amazon mechanical turk, 2005. https://www.mturk.com/.

[96] Hostinger. 000webhost: Free web hosting, 2014. https://www.000webhost.com/migrate?static=true.

[97] The Windows Club. Web browser automatically adds www to url. 2016. https://www.thewindowsclub.com/browser-automatically-adds-www-to-url.

[98] Amber Van Der Heijden and Luca Allodi. Cognitive triaging of phishing attacks. In *28th USENIX Security Symposium*, pages 1309–1326, 2019.

[99] Junjie Zhang, Christian Seifert, Jack W Stokes, and Wenke Lee. Arrow: Generating signatures to detect drive-by downloads. In *Proceedings of the 20th international conference on World wide web*, pages 187–196. ACM, 2011.

[100] Kevin Coogan, Gen Lu, and Saumya Debray. Deobfuscation of virtualization-obfuscated software: a semantics-based approach. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 275–284, 2011.

[101] Martina Lindorfer, Alessandro Di Federico, Federico Maggi, Paolo Milani Comparetti, and Stefano Zanero. Lines of malicious code: insights into the malicious software industry. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 349–358, 2012.

[102] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. Semantics-aware android malware classification using weighted contextual api dependency graphs. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1105–1116, 2014.

[103] Madhusudhanan Chandrasekaran, Krishnan Narayanan, and Shambhu Upadhyaya. Phishing email detection based on structural properties. In *NYS cyber security conference*, volume 3. Albany, New York, 2006.

[104] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *Proceedings of the 29th USENIX Security Symposium*, 2020.

[105] Smartscreen: Report a website.

[106] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 133–144. ACM, 2013.

[107] APWG. Phishing Activity Trends Report 4th Quarter 2020. 2020. https://docs.apwg.org/reports/apwg_trends_report_q4_2020.pdf.

[108] Penghui Zhang, Adam Oest, Haehyun Cho, Zhibo Sun, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kpravelos, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. CrawlPhish: Large-scale Analysis of Client-side Cloaking Techniques in Phishing. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (Oakland)*, San Francisco, CA, May 2021.

[109] www.linkresearchtools.com. Link redirect trace, 2021. https://chrome.google.com/webstore/detail/link-redirect-trace/nnpljppamoaalgkieeciijbcccohlpoh?hl=en.

[110] Phishunt. Exposing phishing kits seen from phishunt.io. https://github.com/danlopgom/phishing_kits.

[111] Sidharth Chhabra, Anupama Aggarwal, Fabricio Benevenuto, and Ponnurangam Kumaraguru. Phi. sh/$ ocial: the phishing landscape through short urls. In *Proceedings of the 8th Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference*, pages 92–101. ACM, 2011.

[112] Sophie Le Page, Guy-Vincent Jourdan, Gregor V Bochmann, Jason Flood, and Iosif-Viorel Onut. Using url shorteners to compare phishing and malware attacks. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pages 1–13. IEEE, 2018.

[113] Bitly. I've found a bitly link that directs to spam, what should i do?, July 2020. https://support.bitly.com/hc/en-us/articles/231247908-I-ve-found-a-Bitly-link-that-directs-to-spam-what-should-I-do-.

[114] Rebrandly. Anti spam policy, 2021. https://www.rebrandly.com/anti-spam.

[115] statcounter. Browser market share, 2020. https://gs.statcounter.com/browser-market-share.

[116] urlscan.io competitors and alternatives. https://www.similarweb.com/website/urlscan.io/competitors/.

[117] Common crawl. https://commoncrawl.org/the-data/get-started/.

[118] ContentKing Academy. Redirect chains: why are they bad for seo?, 2020. https://www.contentkingapp.com/academy/redirects/faq/redirect-chains/.

[119] Gabriela Vatu. Google safebrowsing wrongly blocked bitly links on firefox and chrome, 2014. https://news.softpedia.com/news/Google-Safebrowsing-Wrongly-Blocked-Bitly-Links-on-Firefox-and-Chrome-463132.shtml.

[120] Neha Gupta, Anupama Aggarwal, and Ponnurangam Kumaraguru. bit. ly/malicious: Deep dive into short url based e-crime detection. In *2014 APWG Symposium on Electronic Crime Research (eCrime)*, pages 14–24. IEEE, 2014.

[121] Qian Cui, Guy-Vincent Jourdan, Gregor V Bochmann, Russell Couturier, and Iosif-Viorel Onut. Tracking phishing attacks over time. In *Proceedings of the 26th International Conference on World Wide Web*, pages 667–676, 2017.

[122] KAROL KROL. 10 best domain registrars, 2021. `https://websitesetup.org/choosing-best-domain-registrar/`.

[123] Zhibo Sun, Adam Oest, Penghui Zhang, Carlos E Rubio-Medrano, Tiffany Bao, Ruoyu Wang, Ziming Zhao, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. Having Your Cake and Eating It: An Analysis of Concession-Abuse-as-a-Service. In *In Proceedings of the 30th USENIX Security Symposium (USENIX)*, 2021.

[124] abuse.ch. Measuring reaction time of abuse desks, 2018. `https://abuse.ch/blog/measuring-reaction-time-of-abuse-desks/`.

[125] GRAHAM CLULEY. Phishing attacks exploit youtube redirects to catch the unwary, 2020. `https://www.tripwire.com/state-of-security/featured/phishing-attacks-exploit-youtube-redirects/`.

[126] Asfiya Shaikh. Open redirects (unvalidated redirects and forwards), 2019. https://medium.com/@asfiyashaikh10/unvalidated-redirects-and-forwards-open-redirects-a30d445dd1b0.

[127] CVE. Open redirect vulnerabilities in cve list, 2021. `https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=open+redirect`.

[128] Rapid7. What's going on in production application security 2018, 2018. https://blog.rapid7.com/2018/08/22/whats-going-on-in-production-application-security-2018/.

[129] Google. Suspicious site reporter, 2020. `https://github.com/chromium/suspicious-site-reporter`.

[130] Chromium. The chromium projects - safe browsing, 2020. `https://www.chromium.org/developers/design-documents/safebrowsing`.

[131] Chris Grier, Kurt Thomas, Vern Paxson, and Michael Zhang. @spam: the underground on 140 characters or less. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 27–37, 2010.

[132] Federico Maggi, Alessandro Frossi, Stefano Zanero, Gianluca Stringhini, Brett Stone-Gross, Christopher Kruegel, and Giovanni Vigna. Two years of short urls internet measurement: security threats and countermeasures. In *proceedings of the 22nd international conference on World Wide Web*, pages 861–872, 2013.

[133] Krishna Bhargrava, Douglas Brewer, and Kang Li. A study of url redirection indicating spam. *CEAS (July 2009)*, 2009.

[134] Kumar Chellapilla and Alexey Maykov. A taxonomy of javascript redirection spam. In *Proceedings of the 3rd international workshop on Adversarial information retrieval on the web*, pages 81–88, 2007.

[135] Steve Webb, James Caverlee, and Calton Pu. Introducing the webb spam corpus: Using email spam to identify web spam automatically. In *CEAS*, 2006.

[136] Ian Fette, Norman Sadeh, and Anthony Tomasic. Learning to detect phishing emails. In *Proceedings of the 16th international conference on World Wide Web*, pages 649–656. ACM, 2007.

[137] Se Yeong Jeong, Yun Sing Koh, and Gillian Dobbie. Phishing detection on twitter streams. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 141–153. Springer, 2016.

[138] Sangho Lee and Jong Kim. Warningbird: A near real-time detection system for suspicious urls in twitter stream. *IEEE transactions on dependable and secure computing*, 10(3):183–195, 2013.

[139] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended accounts in retrospect: an analysis of twitter spam. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, pages 243–258, 2011.

[140] Collin Jackson, Andrew Bortz, Dan Boneh, and John C Mitchell. Protecting browser state from web privacy attacks. In *Proceedings of the 15th international conference on World Wide Web*, pages 737–744, 2006.

[141] Gertjan Franken, Tom Van Goethem, and Wouter Joosen. Exposing cookie policy flaws through an extensive evaluation of browsers and their extensions. *IEEE Security & Privacy*, 17(4):25–34, 2019.

[142] Peter Snyder and Chris Kanich. No please, after you: Detecting fraud in affiliate marketing networks. In *WEIS*, 2015.

[143] Neha Chachra, Stefan Savage, and Geoffrey M Voelker. Affiliate crookies: Characterizing affiliate marketing abuse. In *Proceedings of the 2015 Internet Measurement Conference*, pages 41–47, 2015.

[144] Tech Blog (wh). Most Common User Agents, 2012. `https://techblog.willshouse.com/2012/01/03/most-common-user-agents/`.

[145] Privacy Policies. #1 Privacy Policy Generator - Privacy Policies , 2021. https://www.privacypolicies.com/.

[146] Foy Shiver. Apwg and the ecrime exchange: A member network providing collaborative threat data sharing, 2016. https://www.first.org/resources/papers/valencia2017/shiver-foy_slides.pdf.

[147] Amazon. Alexa Top Sites, 2021. http://s3.amazonaws.com/alexa-static/top-1m.csv.zip.

[148] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 319–333. IEEE, 2017.

[149] Cisco Talos. IP & Domain Reputation Center, 2021. https://www.cisco.com/c/en/us/products/security/talos.html.

[150] Danny Cork. A Python package for retrieving WHOIS information of domains., 2021. https://github.com/DannyCork/python-whois.

[151] WEBrate. Webrate.org - Rate the web, 2022. https://webrate.org/.

[152] Peng Peng, Chao Xu, Luke Quinn, Hang Hu, Bimal Viswanath, and Gang Wang. What happens after you leak your password: Understanding credential sharing on phishing sites. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, pages 181–192, 2019.

[153] Alina Oprea, Zhou Li, Robin Norris, and Kevin Bowers. Made: Security analytics for enterprise threat detection. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 124–136, 2018.

[154] Arya Renjan, Karuna Pande Joshi, Sandeep Nair Narayanan, and Anupam Joshi. Dabr: Dynamic attribute-based reputation scoring for malicious ip address detection. In *2018 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 64–69. IEEE, 2018.

[155] Peter Snyder, Cynthia Taylor, and Chris Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 179–194, 2017.

[156] Erik Trickel, Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupé. Everyone is different: Client-side diversification for defending against extension fingerprinting. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1679–1696, 2019.

[157] Stephan Wiefling, Nils Gruschka, and Luigi Lo Iacono. Even Turing Should Sometimes Not Be Able To Tell: Mimicking Humanoid Usage Behavior for Exploratory

Studies of Online Services. In *24th Nordic Conference on Secure IT Systems (Nord-Sec 2019)*, volume 11875 of *Lecture Notes in Computer Science*, pages 188–203. Springer Nature, November 2019.

[158] DataDome. Web scraping protection: How to protect your website against crawler and scraper bots., 2019. https://datadome.co/bot-management-protection/scraper-crawler-bots-how-to-protect-your-website-against-intensive-scraping/#2.

[159] wordpress.org. Wordpress source code., 2022. https://github.com/WordPress/WordPress.

[160] Adam Oest, Penghui Zhang, Brad Wardman, Eric Nunes, Jakub Burgis, Ali Zand, Kurt Thomas, Adam Doupé, and Gail-Joon Ahn. Sunrise to sunset: Analyzing the end-to-end life cycle and effectiveness of phishing attacks at scale. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020.

[161] Yun Lin, Ruofan Liu, Dinil Mon Divakaran, Jun Yang Ng, Qing Zhou Chan, Yiwen Lu, Yuxuan Si, Fan Zhang, and Jin Song Dong. Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[162] Adam Oest, Yeganeh Safaei, Penghui Zhang, Brad Wardman, Kevin Tyers, Yan Shoshitaishvili, and Adam Doupé. Phishtime: Continuous longitudinal measurement of the effectiveness of anti-phishing blacklists. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 379–396, 2020.

[163] radware bot manager. How CAPTCHA Is Used To Block Bots, And Why We Do Not Recommend Using It, 2021. https://www.radwarebotmanager.com/when-to-use-and-when-not-to-use-captcha/.

[164] Suphannee Sivakorn, Jason Polakis, and Angelos D Keromytis. I'm not a human: Breaking the google recaptcha. *Black Hat*, pages 1–12, 2016.

[165] Fabian Stark, Caner Hazırbas, Rudolph Triebel, and Daniel Cremers. Captcha recognition with active deep learning. In *Workshop new challenges in neural computation*, volume 2015, page 94. Citeseer, 2015.

[166] Anh Le, Athina Markopoulou, and Michalis Faloutsos. Phishdef: Url names say it all. In *2011 Proceedings IEEE INFOCOM*, pages 191–195. IEEE, 2011.

[167] Mahmoud Khonji, Youssef Iraqi, and Andrew Jones. Enhancing phishing e-mail classifiers: A lexical url analysis approach. *International Journal for Information Security Research (IJISR)*, 2(1/2):40, 2012.

[168] Mohammed Nazim Feroz and Susan Mengel. Phishing url detection using url ranking. In *2015 ieee international congress on big data*, pages 635–638. IEEE, 2015.

[169] Matthew Dunlop, Stephen Groat, and David Shelly. Goldphish: Using images for content-based phishing analysis. In *2010 Fifth international conference on internet monitoring and protection*, pages 123–128. IEEE, 2010.

[170] Hugo Bijmans, Tim Booij, Anneke Schwedersky, Aria Nedgabat, and Rolf van Wegberg. Catching phishers by their bait: Investigating the dutch phishing landscape through phishing kit detection. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 3757–3774, 2021.

[171] World Economic Forum. The global risks report 2018, 13th edition, 2018. http://www3.weforum.org/docs/WEF_GRR18_Report.pdf.

[172] OP-TEE. OP-TEE Trusted OS Documentation, 2018. https://www.op-tee.org/.

[173] Anthony Lineberry. Malicious code injection via/dev/mem. *Black Hat Europe*, page 11, 2009.

[174] MITRE. CVE-2018-1068 Detail, 2018. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-1068.

[175] Reena Panda, Shuang Song, Joseph Dean, and Lizy K John. Wait of a decade: Did spec cpu 2017 broaden the performance horizon? In *Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 271–282, Vienna, Austria, 2018.

[176] Pavitra Shankdhar. 22 Popular Computer Forensics Tools [Updated for 2018], 2018. https://resources.infosecinstitute.com/computer-forensics-tools/#gref.

[177] VOLATILITY FOUNDATION. Volatility Framework - Volatile memory extraction utility framework, 2017. https://github.com/volatilityfoundation/volatility.

[178] Udo Steinberg and Bernhard Kauer. Nova: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European Conference on Computer Systems (EuroSys)*, pages 209–222. ACM, 2010.

[179] Rui Chang, Liehui Jiang, Wenzhi Chen, Yang Xiang, Yuxia Cheng, and Abdulhameed Alelaiwi. Mipe: a practical memory integrity protection method in a trusted execution environment. *Cluster Computing*, 20(2):1075–1087, 2017.

[180] He Sun, Kun Sun, Yuewu Wang, Jiwu Jing, and Sushil Jajodia. Trustdump: Reliable memory acquisition on smartphones. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS)*, pages 202–218, Wroclaw, Poland, 2014.

[181] Nick L Petroni Jr, Timothy Fraser, AAron Walters, and William A Arbaugh. An architecture for specification-based detection of semantic integrity violations in kernel dynamic data. In *Proceedings of the 15th USENIX Security Symposium (Security)*, pages 289–304, Vancouver, Canada, 2006.

[182] Matthias Lange, Steffen Liebergeld, Adam Lackorzynski, Alexander Warg, and Michael Peter. L4android: a generic operating system framework for secure smartphones. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM)*, pages 39–50, Chicago, IL, 2011.

[183] Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. Flicker: An execution infrastructure for tcb minimization. In *Proceedings of the 3rd European Conference on Computer Systems (EuroSys)*, pages 315–328, Glasgow, Scotland UK, 2008.

[184] Lucas Davi, Patrick Koeberl, and Ahmad-Reza Sadeghi. Hardware-assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software exploitation. In *Proceedings of the 51st Annual Design Automation Conference*, San Francisco, CA, 2014.

[185] Lucas Davi, Matthias Hanreich, Debayan Paul, Ahmad-Reza Sadeghi, Patrick Koeberl, Dean Sullivan, Orlando Arias, and Yier Jin. Hafix: Hardware-assisted flow integrity extension. In *Proceedings of the 52nd Annual Design Automation Conference*, San Francisco, CA, 2015.

[186] Daniel Plastina, Jonathan Cain, and Michael Novak. Methods, systems, and computer-readable media for generating an ordered list of one or more media items, March 25 2005. US Patent App. 11/089,696.

[187] White Paper: An Overview of the Samsung Knox Platform. Samsung Knox, 2016. `https://kp-cdn.samsungknox.com/df4184593021d7b8fabfdfeff5c318ba.pdf`.