2019 IEEE Symposium Series on Computational Intelligence (SSCI)
December 6-9 2019, Xiamen, China

Exploring the Insecurity of Google Account Registration Protocol via Model Checking

Tian Xie, Sihan Wang, Guan-Hua Tu
Department of Computer Science
Michigan State University
East Lansing, Michigan, 48823
Email: xietian1, wangsih3, ghtu@msu.edu

Chi-Yu Li
Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan, 30010
Email: chiyuli@cs.nctu.edu.tw

Xinyu Lei
Department of Computer Science
Michigan State University
East Lansing, Michigan, 48823
Email: leixinyu@msu.edu

***Abstract*—People nowadays use online service accounts (e.g., Google, Facebook, Twitter) to access certain services. Among them, Google accounts have become increasingly important for users. Not only do many Google services (e.g., Gmail, Google Voice, Google Play, etc.) require them, but many online services also trust and rely on them for operational needs (e.g., login based on Google accounts). This trend introduces a new type of attacks that create a large number of fake, but valid, Google accounts. The fake Google accounts allow the adversary to launch various cyber attacks towards Google account-related services. It motivates us to conduct an empirical security study on the Google account registration service. In this paper, we apply model checking techniques to systematically analyze the insecurity of Google account registration service. We develop a model-checking tool, *GAcctAnalyzer*, which consists of two phases: (1) service screening phase, which generates counterexamples from the violation of desired properties, and (2) experimental validation phase, which validates the counterexamples through real experiments. We use *GAcctAnalyzer* to discover four security vulnerabilities including design defects, operational slips, etc. Based on the discovered vulnerabilities, we devise two practical attacks against mobile users and Google: fake Google account generation and Google text/voice spamming attack. They can not only threaten the security of mobile applications and online services, but also cause the Google company to receive user complaints and lawsuits. We finally confirm the feasibility of these attacks through experiments, assess the real-world impact, and propose recommended solutions.**

***Keywords*—Security; Google account; registration; model checking.**

I. INTRODUCTION

Google accounts are essential for users to access many Google services including Gmail, Google Voice, Google Drive, Google Play, etc. Moreover, many online service providers (e.g., eBay, Expedia, New York News, Pinterest, Dropbox, Airbnb, Groupon) allow users to directly access their online services through Google accounts without registering new ones. When the adversary can obtain a great number of Google accounts and abuse them, such attack can threaten not only Google services but also other online services. They may promote malicious and profitable Android applications using fake reviews and downloads, as well as distribute phishing/ad emails, frauds (e.g., eBay seller frauds [1]), and fake news [2]. We believe that the insecurity of Google account registration service/protocol requires more attention from our research community. We thus conduct a study on the sale of Google accounts in the underground economy. We discover that on many websites ([3]–[6]), the Google accounts verified through phone numbers can be sold up to $4.5 each. Besides, there are two findings. First, the sale prices of the US phone-number-verified accounts are eight times more expensive than those of the non-US ones [6]. Second, Google has deployed new security mechanisms against the large-scale account creation and some account creation tools [7] no longer work since 2017.

We then take a security study on how Google defends against the attacks of large-scale account generation. Its security mechanisms mainly focus on user eligibility verification, which verifies whether an account registrant is indeed a real person or not. We find that Google employs four major measures as follows. First, Google prevents a device from being used to register too many (i.e., 10 accounts based on our experiments) accounts by identifying the device's fingerprints. The fingerprints are generated by JavaScript codes in the registration pages. To hinder adversaries from forging valid fingerprints, Google obfuscates the codes. Second, the device fingerprinting mechanism is resistant to the spoofing. Not only can any tampering on a device fingerprint with integrity protection lead a registration request to be rejected, but also a fingerprint cannot be spoofed successfully by the changes of network, software, or hardware settings. Third, Google restricts the phone numbers as well as the times for each phone number to verify Google accounts. The phone number providers are restricted to only mobile network operators (e.g., AT&T). A phone number can be used to pass the verification at most twice per day and ten times during its lifetime. Last, Google employs a proprietary mouse cursor tracker to defend against automated bots.

However, one question arises: "*are these security mechanisms sufficient to secure the Google account registration?*" We observe that people can still buy bulk accounts from the underground markets even though the aforementioned security mechanisms have been deployed. To find the answer, we adopt a model checking technique to systematically examine the registration security. We then develop a tool *GAcctAnalyzer*, which is written in Promela (Process Meta Language, a modeling language [8]), to uncover possible vulnerabilities of the

TABLE I
SUMMARY OF SECURITY VULNERABILITIES FROM THE GOOGLE ACCOUNT REGISTRATION SERVICE.

Vulnerability	Type	Description
V1: One-time Check of Device Usage Limit.	Design defect	The device usage limit (e.g., only 3 accounts without phone number verification can be registered at one device) is checked only once at an intermediate state during the registration process, so a registration instance that passes the check may make the usage exceed the limit after it completes. (Section VI-A)
V2: No or Long Inactivity Timeout.	Design defect	No or long inactivity timeout allows adversaries to make registration instances stay at a certain intermediate state during the registration process and then manipulate them to launch attacks with sufficient time. (Section VI-B)
V3: Loose Limits of Phone Number Verifications.	Design defect	Google places several limits on phone number verifications, but some are too loose and may be abused. (Section VI-C)
V4: Local-view Blockage of Phone Numbers.	Operation slip	Google has only local-view blockage of phone numbers for each individual service. Even if one phone number is blocked by the Google account registration service, it can still be successfully used by other Google services. (Section VI-D)

account registration protocol between the browsers (clients) and Google servers.

GAcctAnalyzer takes two phases: (1) service screening and (2) experimental validation. The first phase generates and analyzes counterexamples from the cases that desired properties are not met, and the second phase validates the counterexamples using real experiments. *GAcctAnalyzer* leads us to discover four security vulnerabilities: one-time check of device usage limit (V1), no or long inactivity timeout (V2), loose limit of phone number verification (V3), and local-view blockage of phone numbers (V4). The first three are design defects, whereas the last one is an operational slip. The details are summarized in Table I.

We devise two proof-of-concept attacks based on the vulnerabilities: (1) fake Google account generation and (2) Google text/voice spamming attack. The first attack allows the adversary to create not only many non-phone-verified accounts (NPVAs), but also phone-number-verified accounts (PVAs) at the cost as low as only 50% of the prices on bulk-account-sale websites. In the second attack, the adversary can spam mobile users with a large number of text messages and voice calls from the Google registration service. It may cause Google to receive user complaints and lawsuits. Note that we seek to disclose new security vulnerabilities of Google account registration, and effective attacks, but not to aggravate the damage. We finally propose recommended solutions to eliminate the vulnerabilities with the goal of minimal impacts on the existing system.

In summary, we make four major contributions as follows.

- We study state-of-the-art security mechanisms of the Google account registration protocol, which are mainly deployed to defend against the attacks of large-scale account generation.
- We develop *GAcctAnalyzer* to explore security vulnerabilities of the Google account registration protocol using a model checker approach. To our best knowledge, we are the first researchers who apply the model checking technique to studying the insecurity of the Google registration service.

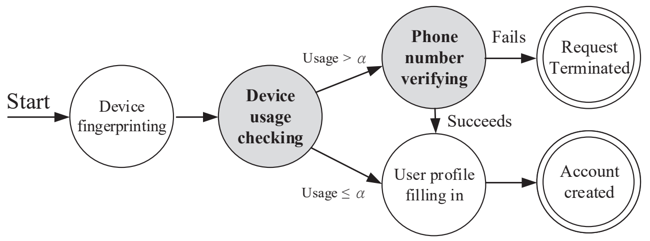

Fig. 1. User eligibility verification for the Google account registration (α is the maximum number of Google accounts that can be created on a single device).

- We devise two proof-of-concept attacks against Google and mobile users by exploiting the vulnerabilities. We assess the real world impact and discuss how they can propagate to mobile applications and online services in practice.
- We propose recommended solutions that only require minimal update of the existing system. They will not only help Google secure its account registration service, but also benefit other online service providers (e.g., Facebook).

II. GOOGLE ACCOUNT REGISTRATION

Some Google services are available only to Google users. A Google account can be registered by filling name, username, password, personal information, recovery email address and agreeing with the privacy the terms of the Google account.

During the registration process, Google verifies user eligibility to prevent fake accounts with two manners as shown in Figure 1. The first manner is to limit the number of Google accounts that can be created daily on a single device, say α ($\alpha = 3$ observed in our experiments). Google identifies devices by their unique fingerprints, which are used to form device IDs. Several techniques have been proposed for this purpose, such as Canvas-based [9] and WebGL-based [10] ones. The device fingerprints are collected through some scripts or applications (e.g., JavaScript codes, Google Chrome browser) at devices while their users access Google websites.

The second manner is to verify users through their phone numbers for the registration. When the number of created accounts reaches α on a day at one device, a new registration request would redirect the user to a "Verify your phone number" webpage. The user can proceed with the registration by verifying the phone number. This can thwart the attack of fake account generation from robots or malicious programs.

III. Threat Model and Methodology

In our threat model, the adversary has no control of Google service infrastructure but his/her own computers. We study the security mechanisms of Google account registration protocol against the large-scale fake account creations and propose *GAcctAnalyzer* to discover the vulnerabilities of the security mechanisms. We thus devise two proof-of-concept attacks based on our findings with the mobile numbers from three major U.S. carriers: Verizon, T-Mobile, and AT&T.

We bear in mind that some feasibility tests and attack evaluations might be harmful to mobile users, carriers, and the service providers. Thus, this study is conducted in a responsible manner through two measures. First, we seek to disclose the vulnerabilities of the Google registration service by our GAcctAnalyzer, but not to aggravate the damage. For example, finite state machines are tested locally instead of brute-force tests on Google directly. Second, in the proof-of-the-concept attacks, we only use our or our colleagues' phones as the victims and do not launch large scale attacks. In addition, we have reported the discovered issues to Google and provided them with our solutions.

IV. Security Study on User Eligibility Verification

In this section, we conduct a security study on the Google's verification of user eligibility. We seek to explore whether the verification can be easily bypassed to generate fake Google accounts. We discover that Google employs four major measures: (1) obfuscated JavaScript codes; (2) anti-spoofing device fingerprints; (3) restricted phone number verification; (4) automated bot-detection. We next elaborate on each of them.

A. Obfuscated JavaScript Codes

We find that device fingerprints are generated by JavaScript codes in the registration page. A registration request cannot be made without enabling the browser's JavaScript support[1]. To secure the generation of device fingerprints, Google makes obfuscation to its JavaScript codes. Specifically, it names both functions and parameters in a meaningless way, and adds many redundant codes. It leads to a very large set of JavaScrpit codes with more than ten thousand lines. Moreover, the codes vary on a daily basis and only partial functions can be observed from the client side. It prevents the adversary from spying how to generate the device fingerprints.

[1]The error message is "*To create a Google Account, turn on JavaScript and try again.*"

flowName: SignUp
gmscoreversion: undefined
deviceinfo: [..., "77185425430.apps.googleusercontent.com", "6b8448e8-0c09-45a7-9f49-0671b380290a", ...]
bgRequest: "web-glif-signup", "!LS6lLg9C9oAl3B9um***qDH5yt04pkJ7q4snnQrqYuxScipeFQ"
f.req: "AEThLlxg***03iwHkw", "Tyler", "Alvin", "Tyler", "Alvin", "TylerAlvin9426", "11pGVXZ14eEmdu", "TylerAlvin9426", true
continue: https://www.google.com/
hl: en
azt: AFoagUUW4npIFKevkE-tIb-iNMY4uHJElg:1537326733528
flowentry: GlifWebSignIn

Fig. 2. A decrypted message that carries encrypted device fingerprints (`deviceinfo`).

B. Anti-spoofing Device Fingerprinting

Our study shows that the device fingerprinting is resistant to spoofing attacks. The generated fingerprints are carried in the `deviceinfo` field of the account registration message, as shown in Figure 2. They have confidentiality and integrity protection. A tempered fingerprint can fail registration requests from its device. We validate the anti-spoofing mechanism by changing environments of a *phone#-verify-required* device and see whether it can spoof the Google registration system to be considered as a new device without the phone number verification. We consider the environments from three dimensions: network, software and hardware. We vary each possible setting while keeping the others the same, and observe whether any verification request of the phone number happens.

Network. We consider two ways of changing the device's IP address. First, we let the device get a different IP address from various campus routers, but they belong to the same autonomous system (AS) number [11]. Second, we let the device get an IP address from different AS numbers by using cellular networks. It is observed that the phone number verification is still required for the device in both ways.

Software. We make three levels of software changes to the test device: different browsers, new OS with virtualization, and new installed OS. First, we do the Google registration using Internet explorer and Firefox instead of Chrome. Second, we install a new OS, Ubuntu 18.04.1 LTS, on a VMWare workstation player, and do registration using Firefox. Third, we replace the original OS, Windows 10, with Ubuntu 18.04.1 LTS. Our result shows that none of these changes can eliminate the phone number verification on the device.

Hardware. We replace several hardware components including CPU, network interface, and memory module on the test device, but the phone number verification is still required. Note that we do not replace its hard disk in this experiment since we want to keep the same network and software settings in the disk.

Finally, we discover that making changes to all the three dimensions together can bypass the check of device finger-

TABLE II
NOT ALL PPHONE NUMBERS ARE ELIGIBLE FOR GOOGLE'S PHONE NUMBER VERIFICATION; IT VARIES WITH TELEPHONY SERVICE PROVIDERS.

Type	Providers	Cost	Supported
Online telephony service providers	Nexmo	$0.57 cents/msg	×
	Google Voice	Free	×
Mobile network operators	AT&T	$30/month	√
	Verizon	$30/month	√
	T-Mobile	$10 cents/msg	√
	US Mobile	$3.75 cents/msg	√

prints of the registration system and thus the phone number verification is not needed. The changes include assigning an IP address from a different AS, installing a new OS, and changing a new motherboard. This mechanism has imposed a large cost on the spoofing attacks which seek to generate fake accounts.

C. Restricted Phone Number Verification

We investigate whether there are any restrictions on the phone number verification from two aspects: telephony service providers and phone number reusability. Our study shows that each of them has some restriction.

Restricted telephony service providers. We study two types of the providers: (1) online providers including Nexmo and Google Voice, and (2) cellular network operators including AT&T, T-Mobile, Verizon, and US Mobile. As shown in Table II, the phone numbers from the first type cannot be used for the verification, but those from the second type can work. Note that when an unsupported phone number is used, the user can receive an error message, "*This phone number cannot be used for verification*."

Restricted phone number reusability. We find that a phone number can be used to pass the verification at most twice per day. When a number has been used to pass two verification tests on a day, the user can receive an error message, "*This phone number has been used too many times*," for further verification tests using the number. We further discover that a number can be used for the verification at most 10 times. Further usage of the number can lead to the error, "*This phone number cannot be used for verification*." Note that this blockage may be permanent. One of our tested phone numbers has been blocked for longer than three months.

Note that Google allows users to receive verification codes from public SMS (Short Message Service) gateways (e.g., www.receive-sms-online.info), which can be used at no cost. It is possibly because their phone numbers are usually provided by cellular network operators.

D. Bot Detection

The adversary may use automated scripts on web pages to invoke some functions by generating button clicks without user involvement. The Google registration process is resistant to this automated bot attack. It employs a proprietary mouse cursor tracker to defend against the bot. Specifically, when a click event is triggered, the JavaScript codes determine whether it indeed comes from the user by checking whether the cursor's coordinate is located within the button's area on the page by obtaining the mouse cursor's coordinate and calculating the cursor's relative position on the screen.

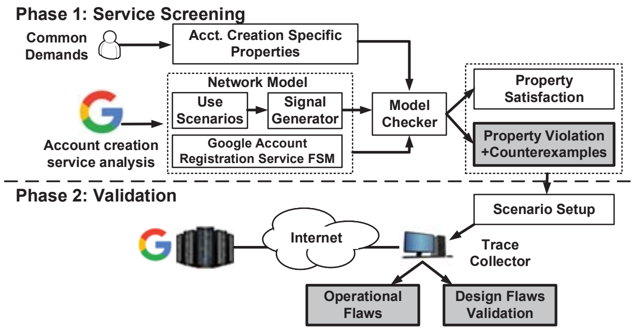

Fig. 3. *GAcctAnalyzer* overview

V. *GAcctAnalyzer*: SECURITY DIAGNOSIS ON GOOGLE ACCOUNT REGISTRATION

Our empirical security study shows that Google has deployed several security mechanisms against the attacks of fake account registration. However, without a rigorous examination based on some formal method, it is unclear whether they are sufficient to secure the registration process. We then develop *GAcctAnalyzer* to conduct security diagnosis on the Google registration. We seek to uncover any design flaws or operational slips from its practice.

GAcctAnalyzer takes a two-phase approach as shown in Figure 3. In Phase 1, it does service screening that explores possible, logical design defects via model checking and then produces corresponding counterexamples. *GAcctAnalyzer* moves to Phase 2, once any design defect is identified. In Phase 2, we validate each counterexample through real experiments.

A. Phase 1: Service Screening

We develop a model-checking tool based on the Spin [8] to discover design issues from the client side. We model states of the Google registration service, and those of client and server interactions. We also define several checking properties specific to the fake account registration (e.g., the maximum number of Google accounts that can be registered). Given all these inputs, we use the model to check whether a set of desired properties are satisfied. For each instance of property violation, which indicates a possible design defect, a counterexample is generated. In this service screening process, there are three issues to be addressed: (1) How to model the Google registration service? (2) How to define the desired properties? (3) How to identify property violations?

1) Modeling: We do modeling from two main aspects: state transitions and usage scenarios.

State transitions. We model both client-side and server-side state transitions for the Google registration service[2]. We build

[2]Our FSMs cannot show all existing state transitions since the site is not controlled by us. However, any identified issues on our FSMs also exist on the real system.

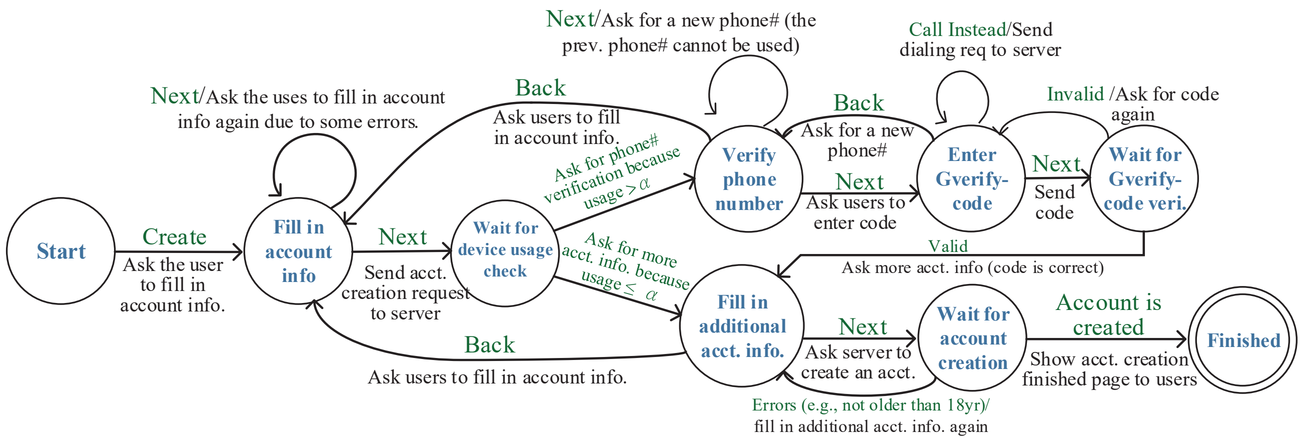
Start
Create
Ask the user to fill in account info.
Fill in account info
Next/Ask the uses to fill in account info again due to some errors.
Back
Ask users to fill in account info.
Next
Send acct. creation request to server
Wait for device usage check
Ask for phone# verification because usage > α
Ask for more acct. info. because usage ≤ α
Next/Ask for a new phone# (the prev. phone# cannot be used)
Verify phone number
Back
Ask for a new phone#
Next
Ask users to enter code
Call Instead/Send dialing req to server
Enter Gverify-code
Next
Send code
Invalid /Ask for code again
Wait for Gverify-code veri.
Valid
Ask more acct. info (code is correct)
Fill in additional acct. info.
Back
Ask users to fill in account info.
Next
Ask server to create an acct.
Wait for account creation
Errors (e.g., not older than 18yr)/ fill in additional acct. info. again
Account is created
Show acct. creation finished page to users
Finished

Fig. 4. The state transition diagram of **client-side** Google account registration service (**blue**: states; **green**: trigger condition, e.g., users click the `Next` button; **black**: the output/action following a state transition).

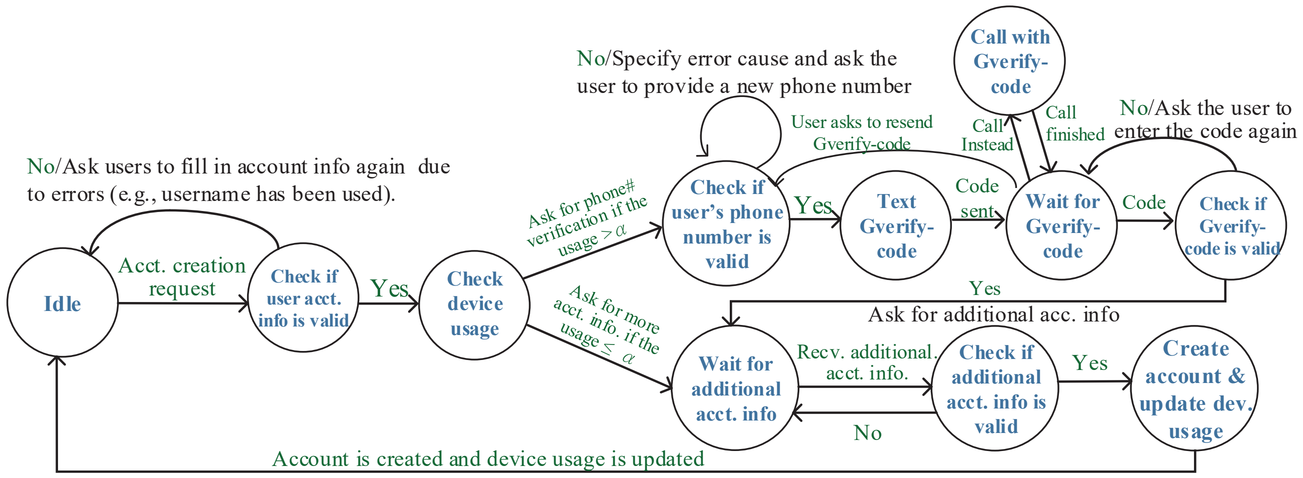
Idle
Acct. creation request
Check if user acct. info is valid
No/Ask users to fill in account info again due to errors (e.g., username has been used).
Yes
Check device usage
Ask for phone# verification if the usage > α
Ask for more acct. info. if the usage ≤ α
No/Specify error cause and ask the user to provide a new phone number
Check if user's phone number is valid
Yes
User asks to resend Gverify-code
Text Gverify-code
Code sent
Call with Gverify-code
Call Instead
Call finished
Wait for Gverify-code
No/Ask the user to enter the code again
Code
Check if Gverify-code is valid
Yes
Ask for additional acc. info
Wait for additional acct. info
Recv. additional acct. info.
Check if additional acct. info is valid
No
Yes
Create account & update dev. usage
Account is created and device usage is updated

Fig. 5. The state transition diagram of **server-side** Google account registration service (**blue**: states; **green**: trigger condition, e.g., receiving user requests; **black**: the output/action following a state transition).

the client-side finite state machine (FSM) shown in Figure 4 by analyzing the Javascript program that validates user inputs and interacts with the Google server from the client. By considering client/server interactions and the client-side FSM, we build the server-side FSM as shown in Figure 5.

Usage scenarios. We seek to explore all the possible client requests and server responses, but it is impractical due to its huge amount of combinations. We thus take a semi-sampling approach as follows. For the components with limited options (e.g., user gender, birthday, and webpage operations), we enumerate all the possible combinations. For the others like user name, phone number, and verification code, we adopt a random sampling method.

We first consider client requests, where a client is allowed to register multiple accounts. We use a run-time signal generator to randomly create user events including `Create`, `Next`, `Cancel`, `Back`, `Resend`, `Call Instead`, and `Waiting`[3]. These events are triggered individually on the registration pages. We then record how the registration service reacts to each event. For example, given a `Create` event at the `Start` state, a new account registration request is created and the service moves to the state, `Fill_in_account_info`. *GAcctAnalyzer* then fills in the input fields of the first registration page with randomly generated account information including first name, last name, username, and password.

We next examine server responses with respect to various client requests. We can observe that the server records the number of accounts which have been successfully created by each user and each verification phone number provided by the user. We equally test all the possibilities, which include rejects with various error reasons. For example, at the `Verify_phone_number` state, there are 6 server error types which can happen for the eligibility verification of a phone number and they are summarized in Table III.

2) Defining Desired Properties: From the client-side and server-side FSMs, we observe that Google mainly relies on verification codes to defend against the attacks of large-scale fake account registration. It shall satisfy two important requirements during the registration. First, in any use scenarios, the number of Google accounts which a user can create without

[3]This event simulates that a user is thinking or waiting at one state.

TABLE III
SIX SERVER ERROR TYPES FOR THE ELIGIBILITY VERIFICATION OF PHONE NUMBERS.

Error Code	Description
ERR1	Please enter a phone number.
ERR2	Please enter a valid phone number.
ERR3	This phone number format is not recognized. Please check the country and number.
ERR4	This phone number has been used too many times.
ERR5	This phone number cannot be used for verification.
ERR6	There was a problem verifying your phone number.

phone number verification should not exceed the designated limit. Second, a phone number cannot be used unlimitedly for the verification.

We define two desired properties for those two requirements respectively: (1) *LimitedAccountCreation_WithoutPhoneVerify*; (2) *RestrictedPhoneNumberUsage*. The first is that the number of registered accounts without phone number verification shall be not greater than a predefined threshold, α. The second is that the times that a phone number can be used for the registration verification shall be not greater than another threshold, β. According to the above study, α and β are respectively set to 3 and 2 per day for one device.

3) Identifying Property Violations: We perform the procedure of the formal model checking. First, the model checker creates entire state space by interleaving all FSMs. We then run the depth-first algorithm to explore state transitions from the initial state under various usage scenarios. Once a property violation is hit, a counterexample is generated. The model checker finally generates all counterexamples, as well as their violated properties and internal states. The internal states include the statuses of all the registration instances, where some of them may have finished the registration and others may stop at intermediate states. We then analyze each counterexample to identify security issues based on its internal states.

Specifically, we mainly focus on two desired properties: *LimitedAccountCreation_WithoutPhoneVerify* and *RestrictedPhoneNumberUsage*. A counterexample of the first property is that more than α accounts without any phone number verification on a trial of one day are generated. That of the second property appears, when the usage times of a phone number for the registration verification are greater than β.

4) Model Checker Implementation: We implement a model-checking tool, *GAcctAnalyzer* in Spin. Specifically, we model the client and server FSMs (as illustrated in Figure 4 and 5) by the modeling language Promela [8]. The *GAcctAnalyzer* runs on a Dell laptop with an installation of Ubuntu 18.04 (CPU: i7-7700HQ, RAM: 16GB). It captures counterexamples caused by the property violations. We further analyze output traces and derive root causes of the violations.

B. Phase 2: Experimental Validation

We set up validation experiments for the generated counterexamples from the service screening phase. We collect plain-text traces of client/server communication and compare them with anticipated results. The experiments cover two popular browsers, Chrome and Firefox, and two computers with Windows 10 and Ubuntu 18.04. We develop a Python script to launch multiple account registration requests. To collect the traces, we deploy an SSL-Split server to intercept and decrypt all HTTPS traffic between the client and the Google server. As for the phone number verification, we use text message services from four major network carriers, AT&T, T-Mobile, Verizon, and US Mobile, and use three smartphone models, Samsung Galaxy S8, iPhone 8, and Google Nexus 6P. The other experimental settings are based on the identified counterexamples in the service screening phase. We also test common use scenarios to explore whether any operational slip that breaks those two properties can be observed.

Note that current *GAcctAnalyzer* has two limitations. First, the implementation of the Google registration service on the server side is not accessible to the public, so *GAcctAnalyzer* may not be able to discover all of its security vulnerabilities. This issue can be addressed by further collaborating with Google, but a non-disclosure agreement may be required. Second, *GAcctAnalyzer* stipulates only properties related to the Google registration in the screening. Other types of security issues have not been covered yet.

VI. SECURITY VULNERABILITIES FROM GOOGLE ACCOUNT REGISTRATION

In this section, we uncover four security vulnerabilities from the Google registration service. The first three are identified via the counterexamples from *GAcctAnalyzer* and can be attributed to design defects, whereas the last one is an operational slip observed from the comparison between the registration and other Google services.

A. V1: One-time Check of Device Usage Limit

We discover that Google checks the device usage limit only once during the registration process, after analyzing the counterexamples caused by the violation of the *LimitedAccountCreation_WithoutPhoneVerify* property. This can be attributed to a design defect. They happen when multiple registration clients are at the states which are subsequent to the `Wait_for_device_usage_check` state, as shown in Figure 4. Even when any clients among them finish registration and cause the usage limit to reach α, the other clients, whose usage limits have been verified, can still proceed to successfully finish the registration without any requests of phone number verification. It means that once the device usage limit is checked while the client-side service transits to the `Fill_in_additional_acct_info` state, it will not be checked again at any subsequent states.

Experimental validation. We validate this vulnerability by showing that $\alpha + 1$ Google accounts (here, it is 4) can be registered at one device without any phone number verification on one day. We first create two Google accounts at our test device, and then let another two clients proceed to finish registration after both of them pass the one-time usage limit check. Thus, four Google accounts are created successfully. We conduct experiments for the latter two clients as follows. First, we create two Chrome browser tabs for those two clients. Second, for both of them, we fill in a form with the basic account information and press the `Next` button. Third, additional account information is requested for both of them. Note that their device usage limits are checked at this moment. Forth, we make them finish the registration one by one. Note that more accounts cannot be registered without phone number verification afterwards.

B. V2: No or Long Inactivity Timeout

Some counterexamples from the violations of the *LimitedAccountCreation_WithoutPhoneVerify* property, show that some registration instances have been staying at the `Fill_in_additional_acct_info` state for a long time (e.g., tens of minutes). We infer that the Google registration process may have no or long inactivity timeout. Though the number of registered accounts has reached $\alpha + 1$ in those counterexamples, these instances, which have passed the device usage check to reach the `Fill_in_additional_acct_info` state, should still be able to successfully create accounts due to V1. It can cause the number of registered accounts to be much larger than α on a day. It can be attributed to another design defect. Note that these counterexamples happen when `Waiting` events keep being generated for those idle registration instances.

Experimental validation. We do validation by letting a registration process stay at the `Fill_in_additional_acct_info` state for various time periods and then proceed to finish the registration. We seek to observe whether any inactivity timeout can lead to registration failure or not. In the experiment, we create four Chrome browser tabs to do Google account registration. For each tab, we proceed to the page with the need of additional account information after entering valid account information. We finish the remaining registration process for these four tabs after 15 mins, 30 mins, 45 mins, and 60 mins, respectively. It is observed that all these registration requests are successful. As a result, the Google registration service has no inactivity timeout or the timeout longer than one hour.

C. V3: Loose Limit of Phone Number Verification

We also get some counterexamples that violate *Restricted-PhoneNumberUsage* property, i.e., a phone number is used more than β times for the verification. After our analysis on their internal states, we discover that a number can be used for the verification more than β times, but only β times can be successful. That is, for one day, a number can continue to be used for the verification until its β-th verification succeeds. It is because an instance can switch back and forth between the `Verify_phone_number` and `Enter_Gverify_code` states. In this case, though the phone number has been used for the verification and a 6-digit verification code is then sent, the instance does not complete the verification with a correct code. The limit is loose for the verification usage and may be abused. We further find that the number of usage times for a phone number on the verification is at most 10.

Experimental validation. We first turn one device to be required for the phone number verification on a new account registration by successfully creating three Google accounts on it. We then use one phone number to do verification for subsequent Google registrations, and manually control the browser to switch back and forth between the `Verify_phone_number` and `Enter_Gverify_code` states by clicking `Next` and `Back` buttons. Each verification trial triggers a 6-digit verification code to be sent out to our phone.

We run this experiment for a week and have the following two observations. First, the 11th verification trial is always denied by Google, so the maximum verification times allowed for a phone number are 10 on a daily basis. Second, the verification times are reset after 24 hours. We validate that a phone number can be used 70 times for the verification in a week. In summary, the usage of a phone number for the verification has three limits: the maximum times for successful verifications per day, those during the phone number's lifetime, and the maximum times for being used for the verification per day. They are respectively 2, 10, and 10. When any limit is met, the verification based on the phone number would be denied.

D. V4: Local-view Blockage of Phone Numbers

We compare the verification limit of the Google account registration with that of other Google services including the activation of a suspended Google account and a Google voice account. We discover that Google does not take the phone number verification as a united function over all of its services; instead, it has only local-view blockage of phone numbers for each individual service. Specifically, even if one phone number is blocked by the Google registration service, it can still work for the other two services. As a result, a phone number can be repetitively used to abuse different Google services.

Experimental validation. We examine whether a phone number that is blocked by the Google account registration can be used to activate a suspended Google account and link a phone number to the Google voice service. We first make an AT&T phone number be blocked by the registration service. By using this number, we are then able to do activation of a suspended account and a Google voice number successfully.

VII. Proof-of-concept Attacks

We devise two proof-of-concept attacks based on the discovered vulnerabilities: fake Google account generation and text/voice spamming attacks. The former's victims are Google, whereas the latter's ones are cellular users and Google.

A. Fake Google Account Generation

We devise this attack by exploiting V1 and V2 and its pseudo code is shown in Algorithm 1. It consists of two steps. First, we start a group of registration instances on one device and let them stop at the

Algorithm 1: Automated Fake Account Registration

```
Input: $N$, the number of accounts to be created
Output: $File$, a file contains the information of $N$ created Google accounts
1  $i = 1; j = 1$;
2  Create $File$;
3  while $i \leq N$ do
4      Create $p_i$;    // open "Create Your Google Account" tab
5      Generate FirstName, LastName, UserName, PassWord;
6      Store UserName, PassWord in $File$;
7      Set FirstName, LastName, UserName, PassWord on $p_i$;
8      Move mouse cursor to "Next" button;
9      Click button; $i++$;
   /* $N$ pages are opened.                                  */
10 while $j \leq N$ do
11     Open $p_i$;
12     Set Birthday $>$ 18-yr old on $p_i$;
13     Set Gender on $p_i$;
14     Move mouse cursor to "Next" button;
15     Click button;
16     $j++$;
   /* $N$ Google accounts are created.                       */
17 Return $File$;
```

`Fill_in_additional_acct_info` state (see Figure 4). Due to V1, all these instances can pass the one-time check of the device usage limit. Second, we make them proceed to finish their registration procedures. Though it may take a little longer time to fill the required information fields at each registration instance, the registration process has no or long inactivity timeout (V2). In this way, each instance can successfully create a Google account without any phone number verification.

Implementation. We implement an automatic tool that uses the Selenium [12] software to control the Firefox browser, instead of an automated script, which is resisted by the Google registration service. At the first step, the tool automatically creates browser tabs, each of which is used to create one fake account. For each tab, it goes to the Google Account creation page and fills in registration forms with fake account information. It then moves the mouse cursor to the `Next` button and presses it. After all the required tabs proceed to the `Fill_in_additional_acct_info` page, the tool starts the second step to finish each tab's registration procedure.

Evaluation. We examine whether our developed tool can indeed create many fake Google accounts successfully. We aim to examine the attack effectiveness instead of aggravating the real world damage, so only 20 fake accounts are considered in this experiment. Our result shows that the tool can first create 20 browser tabs for Google account registration, have them proceed to the `Fill_in_additional_acct_info` page, and then successfully finish their registration procedures one by one. To confirm these fake accounts are valid, we use them to access Gmail and Youtube services without any issue.

Note that the attack costs negligible CPU resource and network bandwidth. But it requires memory to maintain multiple tabs concurrently. Given no inactivity timeout, the bottleneck of this attack is how many Firefox tabs can be concurrently kept on one device. Our experiment shows that each tab takes up to 30 MB memory during the account registration. For a computer with 16 GB RAM, the adversary can create up to 546 fake Google accounts theoretically. One device used to launch the attack before can be used again after 12 days.

Attack variant: low-cost phone-verified accounts (PVAs). The PVAs are the accounts that have passed Google's phone number verification, whereas the fake ones generated by the above attack are non-phone-verified accounts (NPVAs). Google imposes more security restrictions on NPVAs than PVAs. For example, Google does not allow multiple NPVAs to be accessed from the same device. It can temporarily suspend them and ask for phone number verification. But, the PVAs do not have this restriction. The Google PVAs are more expensive than the NPVAs on the markets [3], [4], [6].

We devise this attack variant by leveraging the above attack and V4. It has three steps. First, a cellular phone number ($5.49 from US Mobile with $3.99 one-time SIM card fee and $1.5 sending/receiving 40 texts) with a prepaid text service plan is purchased. This phone number allows us to create 10 PVAs (see V3) and then is permanently blocked by Google for the account registration service. Second, we apply for two Google voice numbers, which require two PVAs and a cellular number. The two PVAs can come from the first step, whereas we can use the same cellular number. Note that one cellular number can be used to apply for at most two Google voice numbers. Due to V4, though the number is blocked for the account registration service, it is still clean for Google voice. Third, we can activate suspended NPVAs to become PVAs by passing their phone number verifications. We use the cellular number and the two Google voice numbers obtained at the second step to activate the suspended NPVAs. Our experiment shows that each number can be used to activate at most 4 suspended NPVAs, so they can totally activate 12 NPVAs to become PVAs. Thus, this attack can create 22 PVAs (10 from the first step) at the cost of $5.49. On the average, the cost of each account is $0.25, which is 50% cheaper than the price $0.5 at bulkpvaseller.us.

Attack incentive and negative real-world impact. Both Google PVAs and NPVAs can be traded on several websites [3], [4], [6]. A PVA averagely costs from $0.5 to $4.5 by considering a transaction of bulk Google accounts. We thus believe that adversaries have incentives to launch a large scale of attack in practice.

The real-world impact of this attack can be negative and far-reaching due to two reasons. First, there are many popular online service providers that allow users to login/access the services with Google accounts, such as eBay, imgur, Wikia, imdb, Pinterest, Zillow, Expedia, Trulia, Realtor, Tripadvisor, to name a few. These service providers share the same security threats with Google and can suffer from a large-scale attack of the fake account registration. Second, in the era of smartphone, adversaries can promote their malicious smartphone applications (e.g., a Bitcoin wallet application accepting the remote BTC transfer commands from adversaries) or services to users as well as generate fake reviews to affect the rating system (e.g., Google Play, Yelp) with the fake Google accounts. The similar service has been seen on the markets [6].

B. Google Text/Voice Spamming Attack

We devise this attack based on V3 to generate text/voice spam messages towards mobile network users. The adversary triggers the spam messages including Google verification codes through the Google account registration service to the users who do not request them. The victims can be both mobile users and Google. The user victims can receive many unsolicited text messages and voice calls from Google, and may pay extra cellular service fees. Google may not only need to pay for those spam messages, but also suffer from possible lawsuits or complaints from the victims.

We develop an attack tool to send text/voice spams as many as possible to victims. The tool keeps a list of victims' phone numbers and generates the allowable spams to them every day. Note that the attack device's IP address will be changed after every 10 spams messages/calls because Google temporarily blocks a device's IP after 10 verification messages. According to our experiment, this mechanism does not have the anti-spoofing device fingerprinting introduced in Section IV-B. Thus, this mechanism can be bypassed by only changing the device's IP address.

Implementation. We implement the attack tool by using the Selenium software to control the Firefox browser. The initial steps similar to the fake account generation attack include creating a new browser tab, accessing the page "*Create your Google Account*" on this tab, filling in a registration form with basic account information, and moving the mouse cursor to the `Next` button to press it. Then it leads the registration process to the phone verification page. It then inputs one victim's phone number, as well as moves the cursor to the `Next`, `Resend`, or `Call Instead` button, presses it and triggers the Google registration service to send the victim a spam text message/voice call for the verification.

The tool repeats the actions on the phone verification page with the attack intervals. After 10 text messages/voice calls are generated to the current victim, a new victim is selected from the victim list to be attacked. On each day, this attack process does not stop until it goes through all the victims on the list. Note that this attack must be launched on a *phone#-verify-required* device (see Section II).

Evaluation. We use 13 phone numbers from our lab members in the attack test. The numbers are from three US major carriers including Verizon, AT&T, and T-Mobile; the residence of participants covers from the East to the West of the U.S. This attack lasts for one week. Our result shows that each tester indeed receives 70 text messages and 70 voice calls from Google. It confirms that a large-scale attack is feasible since this attack is not limited by carriers or victims' locations.

Attack incentive and negative real-world impact. In recent years, there have been several spamming-related lawsuits. For example, Papa John's Pizza faced a $250 million lawsuit for its spam texts in 2012. It finally paid $16.5 million to settle this lawsuit in 2013. In 2018, Bloomingdale paid $1.4 million for a complaint of its spam texts. Thus, victims may accuse Google of the spam texts and voice calls from our developed attack. Google may pay for this spamming attack[4] and lose its reputation.

VIII. Recommended Solutions

We propose three recommended solutions to address the above security vulnerabilities. We seek minimal modifications on the existing system so that Google can eliminate them in a short time.

Atomic registration process. We propose that the Google registration process should be limited to an atomic transaction where the check of device usage limit is done right before the completion of the atomic transaction. Once the number of registered accounts without phone number verification has reached the device usage limit, α, the registration process should ask for phone number verification; otherwise, a new Google account is successfully created. Due to the atomic transaction, the server can process the request with all users' information and then do the check of device usage limit. It can eliminate both V1 and V2, since no intermediate state (e.g., `Fill_in_additional_account_info`) allows a registration instance to stay to pass the limit check without finishing the registration process. Note that the detection of duplicate usernames and emails can be still done while the user fills in input fields without additional intermediate state.

Anti-spam verification. We suggest that Google can take the following two manners to defend against spams generated from the verification service. First, it should reduce the verification limit of a phone number from 10 to a smaller one (e.g., 2 or 3) per day, thereby alleviating the impact of V3. Second, it should provide a way for victims to report the verification spams. For example, the verification text and voice can contain a message that '*G-XXXXX is your Google verification code. If you did not request it, please reply SPAM.*' Google can thus stop an ongoing attack right away.

Unified number blockage system. We recommend that Google should block phone numbers globally with a unified number blockage system. Once a phone number is blocked at one service, this blockage should be propagated to the other services. It is because a number that is used to attack one service may be abused to attack another services later. Google can use a database to maintain the information of blocked phone numbers, and share it with all the services. Such global view of the phone number blockage can eliminate V4.

IX. Related Work

In this section, we present related work in the security aspects of Google and web services.

Google services. There have been several studies that focus on the security of Google services [14]–[18]. Specifically, Gong et al. [14] study security issues of Google+ by developing a model to reproduce its social structure and node attributes.

[4]The company needs to pay for $500 to $1500 per each verified spam text in Miami, FL [13].

Reis et al. [15] discuss key issues about how to shield Google Chrome from attacks, whereas Carlini et al. [16] examine the security architecture of the Google Chrome extensions and perform a security review on a set of 100 Chrome extensions. Zhang et al. [17] report the security analysis of the virtual personal assistants including Google home. Chauhan et al. Lei et al. [18], [19] study the insecurity of using Google home HDVA.

Web services. The security issues of web services are also explored in many papers [20]–[24]. Specifically, Lu et al. [20] study the website fingerprinting and suggest an effective countermeasure to it by removing the information order. Malheiros et al. [21] present a large-scale observation study of user drop-out behaviors on web registration forms from several service providers such as Microsoft and Yahoo. Sengupta et al. Pan et al. [22] analyze the content security policy on real-world websites. Nikiforakis et al. [23] explore the web-based device fingerprinting. Fass et al. [24] propose a detection method of malicious JavaScript codes.

Different from all these studies, our work focuses on the Google account registration service. We systematically explore its insecurity using a model checker approach, and then assess its negative impacts.

X. Conclusion

Google accounts have been considered as important resources. They are required for users to access many Google services. Moreover, due to Google's good reputation, many online services trust Google emails and rely on Google accounts to do resource authorization or user authentication. It can be anticipated that Google services and other online ones can be hurt by fake Google accounts, especially by a large scale of fake ones. Specifically, adversaries can use fake Google accounts to launch various attacks including the distribution of fake news, fake reviews, phishing, spamming, etc. In this work, we study the security mechanisms deployed by Google to defend against the attacks of fake account generation with a model checking tool *GAcctAnalyzer*. We thus find four security vulnerabilities and implement two proof-of-concept attacks. To eliminate the vulnerabilities with minimal impacts on the existing system, three solutions are recommended. We hope that our initial study can attract more attention from the research community and the industry on the insecurity of the Google account registration.

Acknowledgement

This work is supported by the National Science Foundation under Grant Numbers CNS-1815636, CNS-1814551, and by the Ministry of Science and Technology under Grant Number MOST 106-2628-E-009-003-MY3. Note that any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors only and do not necessarily reflect those of the National Science Foundation or the Ministry of Science and Technology.

References

[1] eBay, "Avoiding seller fraud," https://www.ebay.com/help/buying/resolving-issues-sellers/avoiding-seller-fraud?id=4024, 2019.

[2] K. Rogers, "More fake newspaper sites claiming to be based in quebec pop up two years after they were exposed," https://www.cbc.ca/news/technology/quebec-fake-newspapers-1.5228905, 2019.

[3] BulkPVASeller, "Bulk pva seller," bulkpvaseller.us, 2019.

[4] HighQualityPVAs, "High quality pva's online," highqualitypvas.com, 2019.

[5] BuyServiceUSA, "Buy aged gmail accounts," https://www.buyserviceusa.com/product/buy-old-gmail-accounts/, 2019.

[6] "Buy bulk google (gmail) accounts," https://buyaccs.com/en/buy-bulk-gmail-accounts.php, 2019.

[7] "How to create an unlimited amount of phone-verified google accounts (includes gmail, g+ and youtube)," http://www.mytrafficresearch.com/traffic-research/unlimitedgmailaccounts/, 2018.

[8] G. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, 1st ed. Addison-Wesley Professional, 2011.

[9] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *Proceedings of the Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 674–689.

[10] "Webgl - a new dimension for browser exploitation," https://www.contextis.com/en/blog/webgl-a-new-dimension-for-browser-exploitation, 2012.

[11] "Autonomous system number (asn)," https://www.techopedia.com/definition/26871/autonomous-system-number-asn, 2018.

[12] Selenium, "Seleniumhq browser automation," https://www.seleniumhq.org/, 2019.

[13] "Getting spam text messages? you can recover 500−1500 per message," https://www.sflinjuryattorneys.com/spam-text-messages-recover-500-1500-per-message/, 2018.

[14] N. Z. Gong, W. Xu, L. Huang, P. Mittal, E. Stefanov, V. Sekar, and D. Song, "Evolution of social-attribute networks: measurements, modeling, and implications using google+," in *Proceedings of the Internet Measurement Conference (IMC)*. ACM, 2012, pp. 131–144.

[15] C. Reis, A. Barth, and C. Pizano, "Browser security: lessons from google chrome," *Communications of the ACM*, vol. 52, no. 8, pp. 45–49, 2009.

[16] N. Carlini, A. P. Felt, and D. Wagner, "An evaluation of the google chrome extension security architecture." in *USENIX Security Symposium (Usenix Security)*, 2012, pp. 97–111.

[17] N. Zhang, X. Mi, X. Feng, X. Wang, Y. Tian, and F. Qian, "Understanding and mitigating the security risks of voice-controlled third-party skills on amazon alexa and google home," *arXiv preprint arXiv:1805.01525*, 2018.

[18] X. Lei, G.-H. Tu, A. X. Liu, K. Ali, C.-Y. Li, and T. Xie, "The insecurity of home digital voice assistants-amazon alexa as a case study," *arXiv preprint arXiv:1712.03327*, 2017.

[19] X. Lei, G.-H. Tu, A. X. Liu, C.-Y. Li, and T. Xie, "The insecurity of home digital voice assistants-vulnerabilities, attacks and countermeasures," in *IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2018, pp. 1–9.

[20] L. Lu, E.-C. Chang, and M. C. Chan, "Website fingerprinting and identification using ordered feature sequences," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2010, pp. 199–214.

[21] M. Malheiros and S. Preibusch, "Sign-up or give-up: Exploring user drop-out in web service registration," in *Symposium on Usable Privacy and Security (SOUPS)*, 2013.

[22] X. Pan, Y. Cao, S. Liu, Y. Zhou, Y. Chen, and T. Zhou, "Cspautogen: Black-box enforcement of content security policy upon real-world websites," in *Proceedings of the Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 653–665.

[23] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *IEEE symposium on Security and privacy (SP)*. IEEE, 2013, pp. 541–555.

[24] A. Fass, R. P. Krawczyk, M. Backes, and B. Stock, "Jast: Fully syntactic detection of malicious (obfuscated) javascript," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer, 2018, pp. 303–325.