Taken Out of Context: Security Risks with Security Code AutoFill in iOS & macOS

Andreas Gutmann
OneSpan Cambridge Innovation Centre & University College London
andreas.gutmann@onespan.com

Steven J. Murdoch
OneSpan Cambridge Innovation Centre & University College London
s.murdoch@ucl.ac.uk

Abstract

Security Code AutoFill is a new convenience feature integrated into iOS 12 and macOS 10.14, which aims to ease the use of security codes sent via SMS. We report on the first security evaluation of this feature, inspecting its interaction with different types of service and security technologies that send security codes via SMS for authentication and authorisation purposes. We found security risks resulting from the feature hiding salient context information about the SMS message while still relying on users to make security-cautious decisions. Our findings show that adversaries could exploit this decontextualisation. We describe three attack scenarios in which an adversary could leverage this feature to gain unauthorised access to users' online accounts, impersonating them through their instant messengers, and defraud them during online card payments. We discuss the results and suggest possible measures for affected online services to reduce the attack surface by altering the phrasing of their SMS or using alphanumeric security codes. In addition, we explore the design space of Security Code AutoFill and sketch two alternative prototype designs which aim at retaining the improved convenience while empowering users and online services to safeguard their interactions.

1 Introduction

In June 2018, Apple announced at its Apple Worldwide Developers Conference (WWDC) the introduction of a new convenience feature to their operating systems (OS). This new feature, called Security Code AutoFill, scans incoming SMS messages for relevant numeric codes and suggests them to the user for autofill directly on-screen. This improves user experience and convenience as the user is no longer required to open the messaging application, memorise the security code, and re-enter it on another app or website.

Security codes are sent via SMS for a variety of authentication and authorisation purposes. Each requires the user's mobile phone number to be registered with the corresponding service, while the legitimate user is expected to receive and quote security codes to proceed with certain actions – something an impersonator should be unable to do. A malicious exploit of the Security Code AutoFill feature could expose these services and their users to increased risks.

We analysed the interaction between the Security Code AutoFill feature and security procedures of online services that rely on the transmission of security codes via SMS for authentication and authorisation. Our Cognitive Walkthrough analysis found security risks stemming from a design aspect of Security Code AutoFill: reducing the information provided to users while relying on them to make cautious decisions about security. We then explored the design space for this feature and found that alternative designs, which would alleviate our concerns, could be feasible.

Who Are You?! Adventures in Authentication (WAY) 2019.
August 11, 2019, Santa Clara, CA, USA.

2 Background

In this section, we describe the different purposes of security codes sent via SMS and how Security Code AutoFill works.

2.1 SMS-based security codes

One Time Password (OTP) Many systems rely on OTPs for increased security during user authentication. Such security messages are typically sent via SMS to the user's device when registering a login attempt for the corresponding account. To complete the login, the legitimate user is expected to retrieve this code from their device and quote it – something an impersonator would be unable to do.

One Time Authorisation (OTA) Another use for security messages via SMS is delivery of OTA for registration or activation of software installations during setup, e.g. for mobile phone apps. Activation codes can ease such processes and create a permanent link between an installation and a phone number, which can double-function as a unique user ID for services such as instant messengers.

Transaction Authorisation Number (TAN) Transaction Authorisation is commonly used for online transactions to validate that instructions received by a payment provider match with the intentions of the legitimate user of the affected account. It is an essential requirement to defend against sophisticated adversaries [2]. One of the most common methods is to summarise the salient information of a transaction request, augment it with a TAN cryptographically linked to this transaction, and send both via SMS to the device of the corresponding account owner. The user is expected to verify the transaction summary and, if it matches their intention, authorise the transaction by quoting the TAN.

2.2 Security Code AutoFill

Security Code AutoFill works by scanning incoming SMS for numeric codes and keywords, e.g. '*code*' or '*codeword*'. It also monitors the Safari browser and active apps for places to fill-in security codes. If a suitable SMS and input field were found, the feature extracts the security code from the SMS and suggests it for autofill (see fig. 1a). If the SMS contains an amount of money, e.g. '*£100*', this information is displayed in brackets (see fig. 1b). The user needs to tap on this suggestion to autofill the code. This feature operates on iOS and macOS, but requires SMS synchronisation to be activated for the latter.

(a) iOS AutoFill suggestion for a security code.

(b) iOS AutoFill suggestion for a security code from an SMS that also contained the text "£100".

Figure 1: Screenshots: Security Code AutoFill on iOS 12 suggesting autofill of security codes.

2.3 Design of security messages

The design of messages with which security protocols should communicate critical information was first discussed by Abadi and Needham [1], who proposed the principle of 'Explicit Communication': "Every message should say what it means: the interpretation of the message should depend only on its content." Laughery and Wogalter [5] were concerned with the more broad topic of designing general warning messages and recommended to be concise but clearly convey the message, using concrete rather than abstract wording, and avoid unfamiliar abbreviations or ambiguous statements. Short sentences with short, familiar words should be used preferentially. Messages should be explicit in what the reader should do or not do.

3 Threat model

We investigated whether the introduction of Security Code AutoFill has affected pre-existing threats against the affected services, i.e. whether it ameliorates or aggravates protection against known attacks. The adversary considered in our security analysis is located remotely, e.g. without physical access to the online service, user, or their equipment. It is restricted to the usage of publicly known attacks, e.g. no zero-day vulnerabilities. Its active social engineering is limited to sending phishing emails to the user, e.g. no cold calling. iPhones are not jailbroken, e.g. apps or malware could not access and read SMS. We assume that telecommunication providers and their networks are secure, e.g. no SIM swap scams.

4 Methodology

We analysed the interaction of Security Code AutoFill in iOS 12 and macOS 10.14 with security codes delivered by SMS, utilising the Cognitive Walkthrough (CW) method described by Gutmann and Warner [3] for an exploratory data collection to gain insights into how users may perceive the new user interface (UI) and functionality, and the impact this could have on security during interactions with affected services.

This method was selected for its strength at focused evaluations of selected features of a system, while removing the need for partial disclosure about the security focus of our research to avoid participant priming. We began the security evaluation during Apple's *beta* of iOS 12.[1] This required us to be flexible and adapt when a new beta version changed the Security Code AutoFill feature. The choice of an expert evaluation methodology, without direct involvement of non-expert users, minimises the effort for expected repetitions of previously completed tests due to the release of a new beta version. To avoid priming of participants during a user study, we might have been required to deceive them about the actual purpose of the study. Principle 8.07 of the 'APA Ethical Principles of Psychologists and Code of Conduct' [6] about the use of deception in research requires that any effective

[1] Given the expected distribution of new iOS versions to millions of consumer device, we considered a timely security evaluation, prior to the features full release, the ethical choice.

non-deceptive alternative procedures have been excluded as not feasible prior to the use of any deceptive techniques. We concluded that, under consideration of the specific context of this security evaluation, a CW would be a feasible alternative.

The first author conducted the CW and evidenced all findings with screenshots and handwritten notes, which were consequently rewritten into detailed descriptions. These findings were then discussed with and verified by the second author.

4.1 Preparations

We defined the main context, i.e. target of the evaluation, as the iOS 12 and macOS 10.14 UI. The user's main goal during each CW is to complete the task which eventually causes the service to send a security code via SMS, e.g. to login to their remote account. The user's secondary goal is security, e.g. not allowing an adversary to login to their remote account. Their necessary sequence of actions is as follows: (1) proceed with the activity which eventually causes the server to send a security code to their mobile phone, (2) locate the correct input field for the security code, (3) retrieve the security code, (4) conduct security checks as necessary and abort if one or more checks fail, and (5) insert and submit the security code in the previously located input field. Finally, we installed and updated iOS 12 and macOS 10.14 on our devices, prepared a second mobile phone to simulate the adversary, and inserted new SIM cards into the mobile phones.

Each CW consisted of a step-by-step analyses of how the UI could guide its user when attempting to execute the necessary sequence of actions for a corresponding task, and how the adversary could try to incorporate this into an attack. At each step of this process, we assessed: (1) Which visual cues are available to the user for the next action and what feedback is provided to them after each action? (2) What actions could an adversary take to get closer to their goal and how could the user foil such an attack at this step?

4.2 Limitations

The CW method does not involve (non-expert) users and the results are solely based on skills and expertise of evaluators. It commonly identifies only a subset of issues for the evaluated system, for which the frequency cannot be estimated. Yet, this does not reduce the validity of the identified issues.

5 Results

In this section, we report on the results from our CW. For each type of security code, we present one scenario, in which the adversary's attack could succeed, as being representative for multiple findings that lead to similar outcomes for both OS. Although we maintained a detailed record of step-by-step actions during each CW, we limit our reporting to salient information about those actions by the user and the adversary which record relevant information to describe the scenario.

5.1 Remote login with OTP

In the scenario described here, an adversary wants to login to an online account of the user, which is secured with 2FA. As a login portal, we chose the PayPal website[2]. The scenario refers to the iOS 12 UI and assumes that the attacker knows the victim's email address and PayPal password (the user's email address equals their user name for this service).

The scenario begins with the adversary sending a phishing email for an unrelated, 'low-risk' website to the user. Herley [4] argues that users are less likely to detect phishing emails of 'low-risk' websites due to changes in the expected cost-benefit ratio. This could also allow more targeted spear phishing. When the user clicks on a link in this email, and visits the phishing website, the adversary is notified by a scripted event on the website. The adversary then attempts to log in to the user's PayPal account, which requires an OTP for user authentication and suggests to send it to the user's registered phone number – which is confirmed by the adversary. This SMS also summarises the source and purpose of the code.

The user receives the SMS from PayPal on their phone. The phishing website triggers activation of the Security Code AutoFill feature, which suggests filling the 2FA code into it. If the user follows the suggestion and submits the form, the website sends the security code to the adversary, which can use this to login to the user's account. Without Security Code AutoFill the user would be required to access and read the SMS, which would enable them to notice that the SMS was sent by a *different sender* than the website they're browsing, and thus avoid the attack (see fig. 2).

5.2 App registration with OTA

This scenario describes an opportunistic trawling attack aimed at hijacking the accounts of users of an app registered to their phone number. We chose the WhatsApp Messenger app for iOS as an example for our scenario.[2]The scenario refers to the iOS 12 UI and assumes that the adversary is capable of a Man-in-the-Middle (MitM) attack on a public WiFi.

The scenario begins with the adversary executing a MitM on a public WiFi, scanning websites for social login buttons (e.g. Facebook, Gmail, etc.) and injecting a fake WhatsApp login button. The user accesses the attacked WiFi, browses the Internet, and eventually loads a website with social login buttons. They decide to try the "new" WhatsApp login button, unaware of its adversarial nature. The website requests the user to enter their phone number for apparent identification, which they submit and is then transmitted to the adversary.

[2]Our results do not indicate any security weaknesses directly associated with this service. It was chosen for illustrative reasons and a different choice would have been possible.

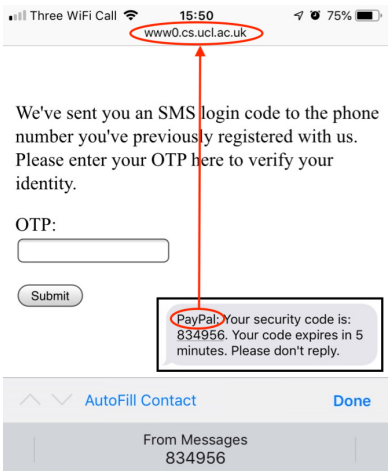


Figure 2: Screenshot: Security Code AutoFill suggested to fill a security code into an unrelated website. The SMS exposes a discrepancy between the SMS content and website.

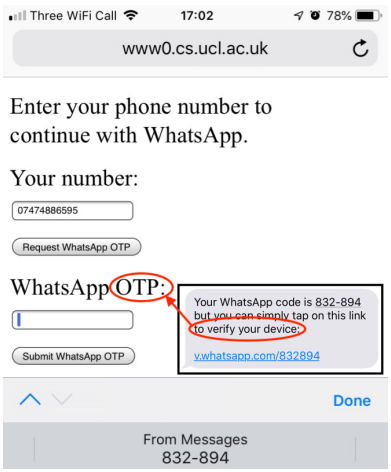


Figure 3: Screenshot: Security Code AutoFill suggested to fill a security code into an unrelated website. The SMS exposes a discrepancy between the SMS content and website.

The adversary installs the WhatsApp Messenger app on their phone and submits the user's mobile phone number during registration. WhatsApp sends an OTA to the phone number which summarises the source and purpose of the code.

The user receives the SMS on their phone. The fake WhatsApp login button triggers activation of Security Code AutoFill, which suggests filling the OTA into this website. If the user follows this suggestion and submits the form, the website sends the OTA to the adversary. The adversary can use this code to register the app on their phone to the user's phone number, hijacking the victim's WhatsApp account. Without Security Code AutoFill the user would be required to access and read the SMS, which would enable them to notice that the SMS was sent for a *different purpose* than that described by the website, and thus avoid the attack (see fig. 3).

5.3 Online payment with Transaction Authorisation

The scenario described here is based on an adversary who wants to trick a user into paying for its purchase. We chose the implementation of 3D Secure with Transaction Authorisation by Monzo Bank Ltd, and the online shops operated by 'Voucher Express' and 'Greater Anglia', as examples for our scenario.[3] The scenario refers to the macOS UI and assumes that the adversary was able to infect the user's device with malware capable of a Man-in-the-Browser (MitB) attack – a common threat assumption for this security technology [2].

The user wants to make a credit card payment of £21.75 at the online shop of merchant Voucher Express. The adversary wants to acquire a train ticket worth £17.30 from merchant Greater Anglia. The user selects to proceed to the payment website of Voucher Express, but is redirected to a payment website for the train ticket by the malware on his device instead. The malware also tampers with the user's browser view to imitate the intended purchase, including an apparent discount to justify the difference in payment value. The user enters their credit card details on this website and requests a security code via SMS. This SMS also summarises the transaction data received by the bank.

The user receives the SMS on their phone. Security Code AutoFill gets triggered and suggests filling the security code into the manipulated website. If the user follows this suggestion and submits the security code, they confirm the purchase of the train ticket for the adversary. Without Security Code AutoFill the user would be required to access and read the SMS, which would enable them to notice that the SMS was sent for a purchase at a *different vendor*, and thus avoid the attack (see fig. 4).

6 Discussion

Our security analysis found elevated security risks from the decontextualisation of security codes by Security Code AutoFill. The attacks we described would be unlikely to succeed if the user were to read the context information in those SMS. We know for more than twenty years that context is critical for security messages [1]. Security codes should not be presented to users without context, but Security Code AutoFill makes any user interaction with such messages optional. We also know that the required contextual information relayed to users in security codes differs based on the type of authentication and authorisation process (see section 2.1), which is why they should not be treated the same way by automated systems that cannot reliably distinguish between them. Services sending

[3] Our results do not indicate any security weaknesses directly associated with these services. They were chosen for illustrative reasons and different choices would have been possible.

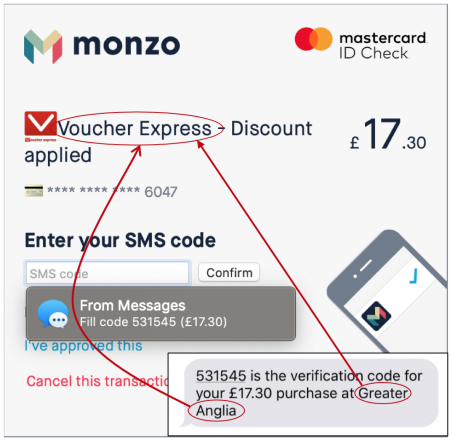


Figure 4: Screenshot: Security Code AutoFill suggested to fill a security code for the wrong purchase into the payment website. The SMS exposes a discrepancy between the SMS content and website / the user's intentions.

security codes should be able to determine whether and how auxiliary software interacts in these processes.

Apple made a unilateral decision to introduce a feature which affects services that rely on SMS to deliver security codes. This might encourage more users to activate corresponding security options for their accounts, but affected services have limited options to influence the feature and how it elevates certain security risks for them and their users. Security Code AutoFill detects numeric security codes based on proximity to words such as "code" or "passcode" and cannot be deactivated[4]. Services could avoid having security codes being recognised by omitting such keywords or when using alphanumeric security codes. But this could be detrimental to user experience and the effectiveness would be subject to future design changes, e.g. introduction of new keywords. Modifications to the website or app code alone would only be a partial solution to prevent the feature from activating, since an adversary capable of content manipulation could reactivate it.

SMS was not designed with support for security protocols in mind and its use therein has been criticised before. Some of the main risks come from the lack of endpoint authentication and cryptographic binding. Solutions that support these mechanisms, e.g. through asymmetric cryptography, have an advantage. Our findings show a further advantage for authentication and authorisation solutions that retain control over interactions with third-party software on the user's side of these processes, e.g. security keys based on FIDO standards.

Services that rely on SMS trade ease of deployment for control over the communication channel. Securing such interactions between independent systems can be a complex and difficult task. Services that wish to utilise benefits from interactions with systems they do not control, while minimising security risks from these interactions, could do so by borrowing techniques from the field of cryptographic proofs: a *reductio ad absurdum*, i.e. *proof by contradiction*, to demonstrate the security of a system. If the existence of a security failure would necessarily require the violation of something assumed to be true, then it implies that no security failure is possible or the assumption is false. One such assumption for systems relying on security codes must be that users can understand the context of each security code before they decide whether to disclose it to an application.

[4]Text Message Forwarding can be deactivated to disable SMS sharing between devices linked to the same Apple ID. Preventing macOS from receiving SMS effectively disables Security Code AutoFill on macOS.

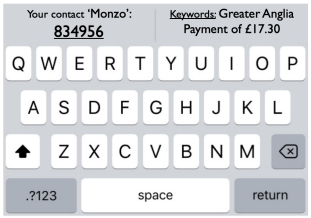
7 Exploring the feature's design space

In this section, we describe two opportunities in the design space of Security Code AutoFill to alleviate the risks described in this paper. We present design sketches and briefly compare their expected functionality with the current implementation of Security Code AutoFill and to a similar functionality in Android. Further work would be required to implement and evaluate actual, functional designs of these sketches, e.g. following a co-design or thinking-aloud methodology.

We identified two main challenges in the design space of Security Code AutoFill: (1) Salient context data shall be extracted from the SMS, yet it shall remain legible for users without the feature, (2) character and space constraints on the length of SMS and from the device's screen, respectively.

The first opportunity aims at displaying more context information from SMS that deliver security codes. The words "From Messages" in current autofill suggestions could be replaced with information about the sender of the SMS, e.g. phone number or contact name. Unused space to the sides of each autofill suggestion could display further context information, if available. To identify such information, additional keywords and design patterns could be specified, e.g. words "sent by" or "purchase at" followed by the desired display name. Similar patterns could apply to other context information such as transaction value and purpose of the code. Figure 5a is a design sketch of such an autofill suggestion. The second opportunity would be to allow SMS senders to identify the targeted destination for a security code, e.g. URL or app name, with specified keywords. The feature would then only suggest codes for autofill into the designated destination (if provided). Figure 5b is a design sketch of such an SMS.

Comparison. Security Code AutoFill was described earlier in section 2.2. Android, on the other side, supports a method for cryptographic binding to specify in the SMS the intended recipient of a security code. Apps can be identified through their cryptographic hash, sent to the remote server when the app requests a security code. The server embeds this hash alongside a security code in an SMS sent to the user's device.



(a) Design sketch of alternative autofill suggestion for an online payment. Sketch of corresponding SMS shown in fig. 5b.

(b) Design sketch of security code sent via SMS with additional keywords indentifying salient context information.

When the SMS is received by the user's device, Android identifies the intended app through its cryptographic hash and makes the message text available to this app through the *SMS Retriever API*. Finally, the app needs to call the *SMS Retriever API*, receive the message text, and parse the security code from it. This provides end-to-end security but only for apps on the receiving mobile device. Security codes intended for websites or apps on other devices, as well as those containing a TAN, need to be processed by the user.

The opportunities for Security Code AutoFill we described could empower users to validate context information of all security codes while retaining the autofill functionality, but would face limitations with respect to the maximum length of SMS. In comparison with Android, our proposal supports all types of security codes intended for any connected device but doesn't fully automate the interaction.

8 Ethical consideration

Security professionals have an ethical obligation to ensure their knowledge is shared in a responsible manner, especially when disclosing risks for systems that have been deployed. In line with our responsible disclosure procedure, we disclosed our concerns during the beta of iOS 12 and informed Apple. An update added transaction values to autofill suggestions (see fig. 1b), which addressed some of our concerns. The issues we report here have not been alleviated. This procedure is compliant with the requirements of our institutions.

9 Conclusion

We analysed Security Code AutoFill in iOS 12 and macOS 10.14 and found security risks stemming from the decontextualisation of security codes, removing salient context information while requiring user's to continue making security-cautious decisions. Our findings show an advantage for security messages delivered directly to users, not only their devices. We described two opportunities in the feature's design space to retain the improved convenience while empowering users and online services to safeguard their interactions, and briefly compared them to the Android platform.

Acknowledgments

This research has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 675730.

References

[1] Martin Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE transactions on Software Engineering*, (1):6–15, 1996.

[2] Manal Adham, Amir Azodi, Yvo Desmedt, and Ioannis Karaolis. How to attack two-factor authentication internet banking. In *International Conference on Financial Cryptography and Data Security*. Springer, 2013.

[3] Andreas Gutmann and Mark Warner. Fight to be forgotten: Exploring the efficacy of data erasure in popular operating systems. In *Privacy Technologies and Policy*, pages 45–58, Cham, 2019. Springer International Publishing.

[4] Cormac Herley. So long, and no thanks for the externalities: the rational rejection of security advice by users. In *Proceedings of the 2009 workshop on New security paradigms workshop*, pages 133–144. ACM, 2009.

[5] KR Laughery and MS Wogalter. Warnings and risk perception. *Handbook of Human Factors and Ergonomics, G. Salvendy (ed.), New York, NY: Wiley-Interscience*, 1997.

[6] American Psychological Association Ethics Committee Washington DC US. APA Ethical Principles of Psychologists and Code of Conduct. *American Psychologist*, 47:1597–1611, 12 1992.