

James Madison University

JMU Scholarly Commons

Masters Theses, 2020-current

The Graduate School

5-2020

Mitigating real-time relay phishing attacks against mobile push notification based two-factor authentication systems

Casey Silver

James Madison University

Follow this and additional works at: <https://commons.lib.jmu.edu/masters202029>



Part of the [Information Security Commons](#)

Recommended Citation

Silver, Casey, "Mitigating real-time relay phishing attacks against mobile push notification based two-factor authentication systems" (2020). *Masters Theses, 2020-current*. 4.

<https://commons.lib.jmu.edu/masters202029/4>

This Thesis is brought to you for free and open access by the The Graduate School at JMU Scholarly Commons. It has been accepted for inclusion in Masters Theses, 2020-current by an authorized administrator of JMU Scholarly Commons. For more information, please contact dc_admin@jmu.edu.

Mitigating Real-Time Relay Phishing Attacks Against Mobile Push Notification Based
Two-Factor Authentication Systems

Casey Lee Silver

A thesis submitted to the Graduate Faculty of

JAMES MADISON UNIVERSITY

In

Partial Fulfillment of the Requirements

for the degree of

Master of Science

Department of Computer Science

May 2020

FACULTY COMMITTEE:

Committee Chair: Dr. Brett C. Tjaden

Committee Members/ Readers:

Dr. Xunhua Wang

Dr. M. Hossain Heydari

Dedication

To Farah, Eva, and Stephanie, who provided me the encouragement and support I needed throughout this journey.

Acknowledgements

I would like to thank Dr. Rick Jones for being a dear friend and inspiring me to take the harder path when available. I would like to thank Dr. Tjaden for guiding me through the thesis process and providing me feedback. I want to thank my parents for all their support and love.

Table of Contents

Dedication	ii
Acknowledgements	iii
List of Tables	vi
List of Figures	vii
Abstract	viii
I. Introduction	1
II. Related Work	6
Existing Protocol Based MITM Protections.....	6
Using Connection Parameters to Detect MITM Attacks.....	8
MITM Protections Integrated into the Application Layer.....	10
Real-Time Phishing Detection.....	11
III. Attack Scenario	14
Simulating a Real-Time Relay Phishing Attack.....	18
IV. PushValidator Implementation	21
Implementation Components	24
PushValidator Service.....	24
PushValidator Demo.....	24
Duo Demo.....	25
PushValidator iOS App.....	25
PushValidator Browser Extension.....	25
VPN & VPN Client.....	26
Tor Browser.....	26
Data Flow Authentication and Integrity	27
Data Points used to Verify the Client Connection	29
Browser Extension Role and Mobile Device Alternatives	31
Message Fields	34
Two-Factor Authentication Request.....	34
Two-Factor Authentication Result.....	35
Push Notification.....	36
Push Notification Response.....	37
Device Registration.....	38
Additional Benefits	39
Pitfalls	40
V. Results	44
Testing Modlishka Against an Application using Duo Two-Factor Push Notifications.....	44
Setup of PushValidator Service.....	50

Testing Modlishka Against Application using PushValidator	52
Testing PushValidator with a Client using a VPN	57
Testing PushValidator with a Client using TOR Browser.....	60
Testing Duo with a Client using a VPN.....	63
Testing Duo with a Client using TOR Browser	65
<i>VI. Conclusion</i>	68
<i>VII. Future Work</i>	69
<i>Glossary</i>	71
<i>Bibliography</i>	72

List of Tables

Table 1 - Two-Factor Authentication Request from Target Application to PushValidator Service	34
Table 2 - Two-Factor Authentication Result returned from PushValidator Service to Target Application.....	35
Table 3 - Content of Push Notification message sent from PushValidator Service to the authenticating user's mobile device	36
Table 4 - Response to the Push Notification sent from the PushValidator iOS App to the PushValidator Service	37
Table 5 - Registration message sent from the PushValidator iOS App to the PushValidator Service	38

List of Figures

Figure 1 - Apple Push Notification Service Architecture [5]	2
Figure 2 - Microsoft Authenticator Two-Factor Push Notification [28].....	17
Figure 3 – PushValidator Request & Response Flow	23
Figure 4 - Device Registration	28
Figure 5 - Duo Demo Application.....	45
Figure 6 - Duo Demo Application Login with Attack in Progress	46
Figure 7 - Duo Demo Application Push Two-Factor Page with Attack in Progress.....	47
Figure 8 - Modlishka MITM Duo Attack Log	48
Figure 9 - Duo iOS App Push Notification with Attack in Progress	49
Figure 10 - Duo Demo Application Successful Login with Attack in Progress	50
Figure 11 - Registering a device in the PushValidator service	51
Figure 12 - Application registration in the PushValidator service	52
Figure 13 - PushValidator Demo Application.....	53
Figure 14 - PushValidator Demo Application with Attack in Progress	54
Figure 15 - PushValidator iOS Push Notification.....	55
Figure 16 - PushValidator Demo Application Two-Factor Push Notification Page with Attack in Progress	56
Figure 17 - PushValidator Demo Application Accessed through a VPN	58
Figure 18 - PushValidator iOS App Push Notification Initiated by User Login through a VPN.....	59
Figure 19 - PushValidator Demo Application Successful Login with User Accessing through a VPN.....	60
Figure 20 - PushValidator Demo Application Two-Factor Page Accessed through the TOR browser	61
Figure 21 - PushValidator iOS App Push Notification Initiated by user Logging in with the TOR browser	62
Figure 22 – QR Code that must be scanned by PushValidator iOS App	62
Figure 23 - Caption showing parsed PushValidator QR code data	63
Figure 24 - Duo Demo Application Two-Factor Page Accessed through a VPN	64
Figure 25 - Duo iOS App Push Notification Initiated by User Logging in through a VPN..	65
Figure 26 - Duo Demo Application Two-Factor Page Accessed from the Tor browser	66
Figure 27 - Duo iOS App Push Notification Initiated by User Logging in with the Tor browser	67

Abstract

This paper explores how existing push notification based two-factor authentication systems are susceptible to real-time man-in-the-middle relay attacks and proposes a system for mitigating such attacks. A fully functional reference system of the proposed mitigation was built and compared to an existing push notification two-factor authentication system while undergoing a real-time man-in-the-middle relay attack. The reference systems used cloud infrastructure for hosting, an Apple iPhone as the notification receiver, and Apple's push notification service to send notifications. A publicly available tool for conducting real-time man-in-the-middle relay attacks was used to conduct the attacks. The results of the tests were recorded and contrasted to show how existing implementations fail to identify such attacks and how the proposed system could. It is recommended that the existing push notification two-factor authentication providers implement additional measures to protect users against real-time man-in-the-middle relay attacks while appropriately weighing key usability issues. While the proposed mitigation system is shown to prevent such attacks, it has usability drawbacks that should be considered.

I. Introduction

The common factors used to authenticate a person's identity are knowledge, possession, inherence and location. These factors are known more colloquially as something you know, something you have, something you are or somewhere you are, respectively. Multi-factor authentication (MFA) uses multiple factors to authenticate an identity [1]. The most widely used MFA is two-factor authentication which uses two factors to authenticate an end user. Most often, the two factors used in these systems are something you know (in the form of a password or PIN) and something you have. The something you have is commonly one of the following: a hardware token which generates one-time passwords (OTPs), a hardware token which is connected to the authenticating computer and transmits the requisite authentication data, a software token such as a mobile app which generates OTPs, a OTP sent via SMS or Email, or a push notification sent to the user's mobile device which the user must choose to accept or decline [2, 3].

Push notification two factor setups are very popular for their ease of use. Like SMS, push notifications are out of band from the authentication flow of the service the user is accessing and available on almost all mobile phones. In a Carnegie Mellon paper exploring two factor adoption among students and faculty at CMU, 91% of survey responders opted for push notifications, 21% used app-generated passcodes and 4% used hard tokens [4]. Push notifications are a common notification pattern for mobile devices and like SMS, their security is largely based on the infrastructure and protocol

used to deliver them to the end device. While there are multiple push notification RFCs [2, 3] in proposed standard status, implementations like Apple Push Notification service (APNS) are proprietary and there is no common, open to review implementation [5]. While most implementations appear to use the well-known quantity of TLS for secure connection establishment, there are other security features and patterns that are not easily reviewable in such proprietary implementations [6].

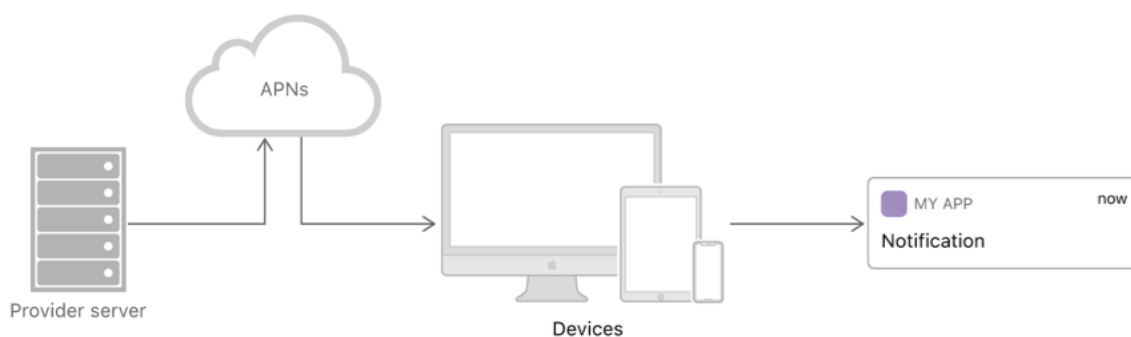


Figure 1 - Apple Push Notification Service Architecture [5]

MFA is a resilient measure to preventing simple phishing attacks which most commonly involves an attacker socially engineering a victim via email to visit an attacker-controlled site where the victim enters a valid set of credentials for a site or service they use. These attacker-controlled sites often mimic the appearance and behavior of the legitimate site or service the attacker is attempting to collect the victim's credentials for. The attacker then is able to use the victim's credentials to login to their account. Based on Verizon's 2019 Data Breach Investigations Report (DBIR), 32% of breaches involved phishing attacks and "was present in 78% of Cyber -Espionage incidents" [7]. Verizon's DBIR report recommends MFA as a first line of defense against such phishing attacks but also cautions that it's not a bullet proof solution. Given that

users who fall victim to phishing attacks believe they are entering their credentials into the legitimate site or server, it should be apparent that the user is also likely to enter their two factor credentials into the phishing site as well if prompted to do so. In the case of a TOTP token the attacker simply needs to enter both the user's username/password combination and the TOTP token within the time the token is valid (normally they're valid in 30 second windows although that window can be elongated to support better usability). While push notifications initially seem more resilient to such an attack since the user doesn't need to enter anything into such a site, the likelihood a user will deny the push notification rather than accept it in a phishing scenario is low. This is because the victim believes the phishing site to be authentic and thus the push notification as well. Additionally, the victim has been conditioned to press accept on the notifications when they're expecting them which means they're less likely to inspect any additional validating information provided by the notification. Such an attack on push notifications or TOTP MFA requires the attacker to be actively passing along the stolen credentials to the legitimate site and executing the authentication flow in order to make use of the TOTP token in a timely manner or to make the push notification appear on the victim's device auspiciously.

Each of the aforementioned possession second factor implementations have their advantages and disadvantages which attempt to balance ease of use, recoverability, and security. OTPs sent via SMS are regarded as the weakest security option because of the weaknesses inherent in cellular networks and provider processes. A recent case of SMS second factor being exploited was a compromise of several Reddit

employee accounts by performing a SIM swap and causing the SMS message containing the OTP to be sent to the attacker's phone [8, 9]. Mobile apps serving as software tokens and hardware tokens which generate time-based one-time passwords (TOTPs) are one of the most popular options because of the ubiquity of mobile phones. TOTPs require the user to manually type them into the system they're authenticating to which makes them susceptible to real time, man-in-the-middle relay phishing attacks. These phishing attacks involve an attacker tricking a victim into entering their two factors into a phishing site which automatically relays the user's credentials to the intended site in real time. This attack renders the two-factor protections useless because the user will believe they're entering their password and second factor into the intended site. The most effective possession factor is the hardware key which transmits the authentication data to the site. These hardware keys use the U2F protocol, and more recently WebAuthn, to communicate with the target application [10, 11]. These protocols mitigate phishing attacks by performing checks to ensure the target site is in fact the one intended and using public key cryptography. In order to compromise these hardware keys, the attacker would have to gain physical access to them which requires a great deal more complexity and risk than the aforementioned attacks. The issue with hardware keys is that support for them is extremely limited. Push notifications are becoming an increasingly popular second factor option because they occur out of band and require the user to accept or decline the attempted access to their account using their mobile device providing a notification in the event the user's credentials were compromised and an attacker attempted to login with them. However, push

notifications are just as susceptible to man-in-the-middle phishing attacks as OTPs for a variety of factors which will be discussed further in future sections.

In this paper we'll show how potential main-in-the-middle relay phishing attacks against pushed based two-factor authentication would work and propose potential mitigations which can be implemented by sites or two-factor providers. Our mitigations will be layered onto existing push notification and web authentication protocols in order to provide users with less friction during authentication, maintain the current level of application development complexity, and reduce the risk of relying on a more untested, custom authentication protocol which requires intense scrutiny before widespread, standardized use.

II. Related Work

There are several research implementations which look to deal with real-time phishing attacks, but few are focused on improving practical, widely used authentication techniques. Push based two factor authentication systems, in particular, have had limited research with respect to the ability to detect real-time MITM phishing attempts via connection metadata and parameters.

Existing Protocol Based MITM Protections

HTTP over TLS/SSL is often regarded as a primary defense against MITM phishing attacks on the internet. However, HTTPS relies heavily on the end user identifying they're visiting a malicious site. Users have been trained to trust sites where the browsers show a green lock symbol, but browsers will show that symbol for any site with a valid certificate signed by a trusted CA. With the rise of free TLS certificate availability, such as through LetsEncrypt [12], such a certificate is trivial to obtain and only requires proving domain ownership. In June of 2019 the FBI released a public service announcement warning that cybercriminals are increasingly using valid TLS certificates for their malicious sites to exploit user's trust in the browser's lock icon [13]. Proposed techniques to deal with HTTPS MITM attacks include using third parties to check the presented certificate against a known good list or using a variation of a PAKE protocol such as DVCert described by Dacosta *et al* [14]. Such techniques present usability challenges for the end user or significant changes in existing application

authentication mechanisms or TLS. In fact, Extended Validation (EV) Certificates, which are supposed to require significantly more checking by Certificate Authorities, have been shown to be very easy to get and are able to be manipulated to appear similar to existing companies. Research by Ian Carrol shows that he was able to obtain an EV certificate for “Stripe, Inc.” a company he had purported to setup for the sole purpose of obtaining the certificate [15]. This certificate would look like the legitimate Stripe’s certificate (Stripe is a common payment processor). This sort of easy manipulation to make a certificate for a fake site look legitimate makes user confusion and mistakes more likely. In a study by Adrienne Porter Felt, *et. al.*, that looked at how existing browser security indicators were interpreted by users, the authors found that over 20% of users felt Chrome’s green lock icon meant the site was secure or “safe to enter data” on [16]. The study found that over 35% of users regarded the website as being a “secure site or page” or “safe” [16]. The trend to exploit such trust is on the rise as well. The 2018 Phishing Trends & Intelligence Report by PhishLabs indicates that one third of all phishing sites have a valid HTTPS certificate [17].

Using Connection Parameters to Detect MITM Attacks

The key challenges of detecting and mitigating real time MITM phishing attacks is that the server needs to validate that the client who is presenting its credential is in fact the expected client and the client needs to validate it is in fact talking to the expected server. A given TLS connection is in practice, not able to be tied to a user other than the authentication occurring at the application level. Rolf Oppliger *et al* explore how to use parameters of the established TLS connection in the authentication flow to prevent successful MITM attacks [18]. The authors propose generating a user access code per TLS session which is created by creating a hash of the TLS handshake messages and signing it with a private key from, preferably, a hardware token [18]. The idea is that the server will be able to determine whether the TLS session that the client used to deliver the credentials to the server is in fact the same one as the one the server received the credentials on. This approach is similar to the *ZeKo* protocol, which will be discussed in more detail later in this chapter, where the IP address of the client is transmitted which provides key information on the communication channel the server should expect to be communicating with. A thesis written by Radi Abubaker, Channel Based Relay Attack Detection Protocol, uses a very similar approach to detecting a relay in a wireless channel communication [19]. The approach, generally described, is to require the secure transmission of unique qualities of the wireless channel itself so that the receiving end can validate the channel it's receiving on is in fact the same the sender is sending on.

Zero knowledge, asymmetric cryptographic authentication systems are another commonly referenced solution for protecting against MITM phishing attempts, real time or otherwise. In the face of a relay MITM attack, these protocols must securely pass the user's network address to the server. Doing this allows the server the opportunity to validate that the connection it has established is in fact the end user and not a MITM. Because the attacker will likely have a different network address (unless both the attacker and victim are behind the same NAT), the server is able to determine the attacker is not the legitimate end user they're expecting to be connected to. The need for the client to securely disclose their IP address to the server is described as a key feature of *ZeKo* by Paul Knickerbocker in *Combating Phishing Through Zero-Knowledge Authentication* [20]. In *ZeKo* the client provides its IP address securely in order for the server to be able detect MITM relay attacks as well as replay attacks. These protocols suffer from similar issues as PAKE solutions for HTTPS in that they have the pitfall of requiring significant changes to existing authentication schemes [20].

Another proposed solution from Italo Dacosta, *et. al.*, proposed a means of using user application credentials with a PAKE protocol to provide the browser the expected certificate fingerprints that domain will use [21]. This allows the browser to securely check the certificate fingerprint against the stored expected fingerprint. If there is a mismatch between the fingerprints a man-in-the-middle is assumed to be occurring and the browser should halt communication [21]. It's important to note the protocol only provides for server authentication as it relates to a particular domain and does not perform user authentication or protection against phishing sites on a different domain.

MITM Protections Integrated into the Application Layer

The reason MITM relay attacks can be successful with little knowledge of the authentication process is because in modern web authentication systems is that web servers inevitably return a session key, often in the form of a cookie, which is used to continuously authenticate with each subsequent request since HTTP is stateless. Obtaining one of these session keys, also known as session hijacking, allows an attacker to impersonate the legitimate user for future requests until the key expires. Since a real time, MITM relay attacker can simply pass valid authentication credentials and responses back and forth between the server and client they only need to wait for the server to present the session key as a cookie which they can acquire and reuse. This means the attacker does not need to actually have any knowledge of the user's credentials as long as neither the client, nor the server, can detect the attacker in the middle prior to the server presenting the authentication cookie and the attacker is able to acquire the cookie in a readable form. The attacker will then be able to use the cookie undetected. Italo Dacosta, *et. al.*, introduced a "one-time cookie" in their paper, One-Time Cookies: Preventing Session Hijacking Attacks with Disposable Credentials [22]. In the paper, a method for one-time use cookies is proposed such that an attacker who hijacks a session will not be able to use the captured cookie or generate the next expected cookie to authenticate with [22]. One of the key strengths of this approach is the use of the URL in the HMAC portion of the one-time cookie. This means that a user visiting a MITM relay phishing site would generate invalid cookies such that an attacker

would not be able to simply relay them to the target legitimate site. This method still would allow a MITM relay to succeed though because if the MITM relay site acquires the user's credentials they can setup a valid session and gain all of the required secret material to generate the one-time cookies because neither the server, nor the client, is able to detect the attack occurring in real time.

Real-Time Phishing Detection

Detecting phishing in real-time is often a difficult task evidenced by the lack of existing techniques for doing so. Most often browsers, such as Google Chrome [23], check domains prior to loading a web page against a blacklist to see if the domain has been identified as a phishing domain. The primary weakness of this approach is that phishing domains are constantly being setup and a new one that has seen little to no traffic is unlikely to have been identified as phishing. Users who are the first to visit a new phishing site will be unlikely to be alerted by their browser that the site is in fact a phishing site.

Two of the most common proposed approaches for real time detection of phishing sites are URL analysis and site content analysis. Since most phishing sites attempt to mimic a legitimate site as closely as possible, they'll often use content by linking directly to the target legitimate site or they'll save a copy of the site assets and render those with small modifications. There are even tools for quickly cloning a site such as *setoolkit* which comes with most distributions of Kali Linux [24]. One approach proposed by Sadia Afroz and Rachel Greenstadt in *PhishZoo: Detecting Phishing*

Websites By Looking at Them, details using the content of a site using computer vision and other techniques to determine how similar it is to a legitimate site [25]. PhishZoo saves profiles of specific sites and when a user visits any site with an unknown URL, that is a site that has not been saved as a profile, PhishZoo compares the unknown site's content to all of the saved profiles to see if there is a match. If there is a match the site is deemed to likely be a phishing site. PhishZoo can be supplemented with existing URL heuristics, blacklist, and whitelist approaches. The approach has challenges due to how dynamic modern sites are as well as how often they are updated. This means the various site profiles have to be updated regularly and a site with too much dynamic content or a volatile layout may result in a high rate of false negatives.

Using heuristics or machine learning techniques to identify phishing URLs is another common technique. This approach uses various features of the URLs and domains to identify them as potential phishing sites. Such an approach is detailed by Jianyi Zhang and Yonghao Wang in *A Real-time Automatic Detection of Phishing URLs* [26]. The approach detailed in the paper uses a logistical regression classifier utilizing lexical features of the URL and whether the site is using a virtual host. Several well-known data sets are used as training sets and the training is run once per day as the training sets are updated. This approach only works when the URL bears the qualities that match the training set features and if the attacker has been able to compromise the victim's DNS infrastructure or internal network this approach will not detect it because the URL will match the legitimate site's. Additionally, new URL patterns can be used by

attackers to evade detection because the approach is highly dependent on the training data set.

Another approach detailed in *Beyond the Lock Icon: Real-time Detection of Phishing Websites Using Public Key Certificates* uses features of website TLS certificates with a machine learning algorithm to detect phishing sites in real time [27]. The features used include the certificate authority name, valid dates, subject name, and extensions. By using a training set of certificates from identified phishing sites and from legitimate sites the classifiers are able to classify new phishing sites. The certificates that are classified as phishing must have certificates with features deemed similar to the phishing certificates in the training set by the classifiers. The authors note that their approach is more robust than URL and content analysis techniques, but the features used are susceptible to attacker manipulation and as a result allow a non-trivial possibility of successful evasion techniques.

III. Attack Scenario

The attack scenario this paper describes involves an attacker tricking a victim user into visiting a malicious site meant to appear as a legitimate site, also known as a phishing attack. Typically, in such a phishing attack, a user enters their credentials into the phishing site which can later be collected and used by an attacker in a manual or automated fashion (such as in a credential stuffing attacks[ref?]). However, if the victim user has two-factor authentication enabled on their account the attacker will not be able to successfully authenticate completely using the phished credentials because they will not have obtained usable two-factor credentials. If the attacker employs a real-time relay attack though, they are able to pass the user's username and password as well as any valid two factor credentials to the target application as needed. The user will be unlikely to notice such an attack because they will be under the impression they're logging in to the desired target application. The user's only way to notice the attack is to notice the URI of the site they're on is not the target application. In many cases this may be difficult for the user to notice because of limited UI real estate on mobile devices or URIs which appear to be very similar to the target application's. Additionally, an attacker can have the malicious site use a TLS certificate which the user's browser will show as valid and secure because all validation checks from the browser's perspective will pass, i.e. the URI matches the alternate name of the certificate, the CA which signed the certificate is trusted by the browser, the certificate has not expired. This leads to a false sense of security from end users because the industry has pushed the lock icon in

browsers to mean the user is “safe” and “secure”. Because so much onus is on the end user to be able to identify when they’re not on the site they intend to be by thoroughly checking the URI, there is ample possibility for even the most technical users to make mistakes. To compound the problem further, when the two-factor authentication system a user has on their account is push notification based, they have even more responsibility to identify malicious logins.

Push based two-factor authentication has a few very strong advantages including convenience, ease of use, and a real time notification of when compromised credentials are used. The downsides, which have not been extensively publicized as of the time of writing, include a habitual behavior to accept such notifications. Statistics from push based two-factor providers are not readily available but it’s reasonable to assume that the vast majority of two-factor push notification requests are accepted because cases of compromised credentials would be relatively rare compared to normal login activity. This means a user has been trained to accept such notifications or otherwise not to inspect them closely to identify malicious logins when they’re expecting them. Another downside is that even for attentive users, aside from timing, their only means to identify a malicious login is to examine the IP address and associated IP’s geolocation if available. This is problematic for a number of reasons, not the least of which is that an average user has no idea what their IP is nor any semblance of what a suspicious value would be. Providing geolocation data on the associated IP is helpful however, it can often be incorrect or confusing causing users to ignore it. For example, it is not uncommon for users on a mobile network to be given an IP which geolocates to a

completely different location than their current location. Additionally, if the user is accessing a resource on an internal network or they're using a service which aggregates connections such as a VPN, their true external IP will be obscured or irrelevant.

Attackers can also host their malicious application on cloud providers which will allow them to use IP addresses which will geolocate to a variety of locations, some of which may be close enough to a user to appear valid for even more technically savvy users.

Some push notification two-factor providers, such as Microsoft, don't even provide the client IP address or the associated geolocation data. This gives the user no opportunity to identify malicious logins. These issues with push notifications play perfectly into making a real-time relay phishing attack scenario viable. Microsoft's push notification is show in Figure 2.

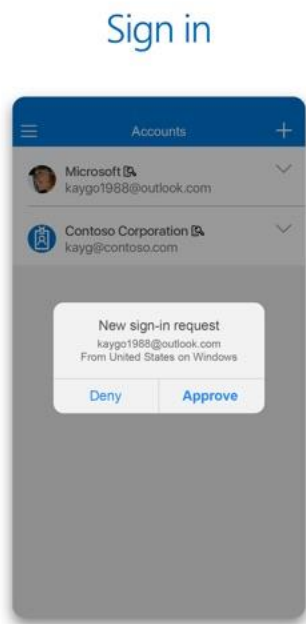


Figure 2 - Microsoft Authenticator Two-Factor Push Notification [28]

When a victim user is visiting a malicious application performing a real-time relay phishing attack, the user enters their credentials believing they're accessing the desired target application. As a result, after the malicious application relays the victim user's credentials to the target application it will send a two-factor push notification to the victim user's mobile device. The user is likely to accept this two-factor authentication request because of the aforementioned reasons around trained behavior and lack of technical expertise. To compound the issue further, the user is unlikely to even suspect the request as malicious because they believe they're logging in to the desired target application and are expecting to receive the push notification.

Simulating a Real-Time Relay Phishing Attack

To simulate the real-time relay phishing attack described previously, two web applications were created to serve as the target applications. One target web application was integrated with Duo Security's two-factor push notification service via their .NET SDK [29]. To login to the application the user must enter their username and password on the login screen. After doing that successfully the web application uses Duo's provided SDK to send a request to the Duo two factor service which then sends a push notification to the user's mobile device. Once the user confirms the push notification the target web application will authenticate the user and allow them to visit authorized resources. The second target web application was integrated with the PushValidator two-factor push notification service which is the novel solution presented by this paper to mitigate the attack scenario previously described. The PushValidator solution is described in more detail in future sections.

In the real-time relay attack scenario, there is a malicious application the user will visit which will masquerade as the target web application. It does this by, in real-time, querying the login page and relaying it to the end user. The user enters their credentials which the malicious application relays to the target application over HTTP as a valid user would. This causes the target web application to send the two-factor push notification to the user's mobile device. If the user accepts this push notification the target web application will authenticate the malicious application and provide it a session cookie for continuous access and the victim user would then be redirected to

the target application or the malicious application could continue to relay between the victim user and the target application. The malicious application in this simulation was created using Modlishka [30] which is a tool built to serve as a reverse proxy for legitimate sites while capturing credentials and cookies by parsing and rewriting the HTTP requests it's proxying.

The target applications were deployed to Azure Web App service [31] and publicly available over the internet. Likewise, Modlishka was deployed on an Azure VM running Debian 9.9 and the domain pushvalidator.com provided by Google Domain service pointed at the VM's public IP to perform the successful proxying and credential capture. Additionally, Modlishka used a valid certificate for pushvalidator.com from LetsEncrypt [32], so that when visiting the site a victim would not be shown any security warnings. Modlishka was configured using a JSON file with options describing the domain pointing to it, the certificate, private key and CA it should use for TLS connections, the target web application it would proxy connections to, and what IP to bind to on the VM. An example configuration, with secrets redacted, is provided below.

```
{
  "proxyDomain": "pushvalidator.com",
  "listeningAddress": "0.0.0.0",
  "target": "pushvalidatordemo.azurewebsites.net",
  "targetResources": "",
  "targetRules": "",
  "terminateTriggers": "",
  "terminateRedirectUrl": "",
  "trackingCookie": "id",
  "trackingParam": "id",
  "jsRules": "",
  "forceHTTPS": false,
  "forceHTTP": false,
  "dynamicMode": false,
  "debug": true,
  "logPostOnly": false,
  "disableSecurity": false,
```

```
"log": "requests.log",  
"plugins": "all",  
"cert": "",  
"certKey": "",  
"certPool": ""  
}
```

IV. PushValidator Implementation

In order to mitigate a real-time relay phishing attack, either the client or server needs to be able to detect that there is a man-in-the-middle proxying the requests and responses. To properly detect this, the server or client must have some way to determine that they are not connected directly to their expected peer. One of the most straightforward approaches to do this is to compare the expected IP of the peer to the IP they're actually connecting to. An example of such an approach can be seen in *ZeKo* by Paul Knickerbocker where the client's IP is provided via the proposed zero knowledge authentication protocol in order to detect a MITM attack [20]. Additional data points can be tested as well to confirm, such as the TLS server certificate fingerprint and URI. Additionally, the actual data points should be conveyed to their peer out of band and need to be tamperproof in order to ensure the man-in-the-middle is not able to modify them to match their current attack position or otherwise remove them.

To implement such a solution for two-factor push notifications it is convenient to use the push notifications and subsequent responses from the user's mobile device, as a medium to transport the actual data points to the target application via a third party two-factor provider. In order to simulate this end-to-end, a full two-factor service was built which consists of a web application a user can log into to register their device for the push notifications and where applications can register which will submit two-factor requests to be sent to the end user. An iOS app was also built which can receive push notifications, gather the actual data points and then submit the two-factor

authentication response along with the data points to the two-factor authentication provider. A browser extension was also built, which serves to collect the actual data points from the user's machine requesting to authenticate to the target web application.

The flow of an authentication attempt is as follows: First a user enters their username and password into the target web application successfully. The target web application generates the payload for the two-factor authentication request containing the user's username, client IP address and HMAC which it returns to the user's browser to submit to the two-factor provider service. The browser (via JavaScript) submits the provided payload to the two-factor provider which then uses the username to find a registered device for that user and sends a push notification to it which contains the client's IP and username. The user receives the push notification on the two-factor provider's mobile application and upon accepting it must scan a QR code which will provide the server's IP, URI and the certificate fingerprint of the web application the user is actually connecting to. The QR code is generated by the aforementioned extension installed in the user's browser that is able to inspect the URI, IP and certificate of the site the user is actually on. The information from the QR code is then submitted back to the two-factor provider via a HTTP request from two-factor provider's mobile application. The user's browser, via JavaScript, retrieves the two-factor authentication result from the two-factor provider directly and then supplies it to the target web application. The target web application is then able to verify the response by comparing the provided HMAC in the response to a HMAC it generates using message fields and

application symmetric key. The IP, URI, and certificate fingerprint the user is connecting to against their own actual values from the response are compared against the server's known values to determine if there is an active MITM occurring. If there is an incongruity then the server is able to deny the authentication and take additional measures to mitigate further compromise of the user's account. These actions could include locking the user's account, sending them an alert via email or mobile push notification, or blacklisting connections from the attacker's MITM IP. Figure 3 provides a detailed flow of the various responses and requests.

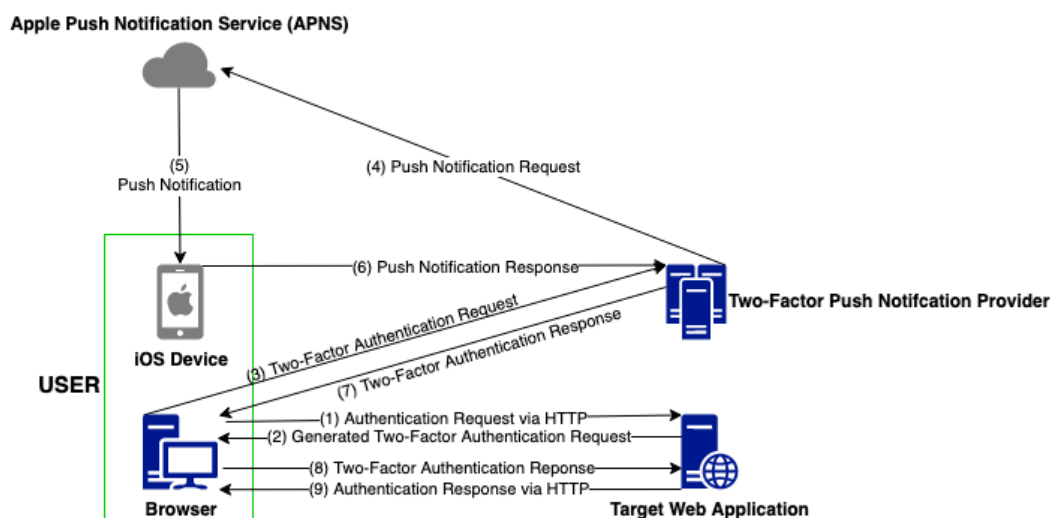


Figure 3 – PushValidator Request & Response Flow

Implementation Components

Below are the individual components making up the PushValidator implementation described in more detail. These details include the associated infrastructure and configurations used to conduct tests and simulate various scenarios. All of the associated code is publicly available via GitHub [33, 34, 35, 36, 37].

PushValidator Service

This application is the two-factor push notification provider. It allows users to register their push notification mobile device using the PushValidator iOS app. It also allows application administrators and developers to register their application on the service. The application is responsible for receiving the two-factor authentication requests from the target applications, sending the two-factor push notification to the user's registered device, and receiving the two-factor authentication result.

The application is written using ASP.NET Core 2.1 and deployed to Azure Linux Web Apps Service. The database used is an Azure SQL database. The configuration options for the application include the database connection string, the credentials needed to send push notifications via the Apple Push Notification Service, and the application logging level. The code is available on GitHub [37].

PushValidator Demo

This application is the demo target web application which uses the PushValidator Service as its two-factor push notification provider. The application is written using

ASP.NET Core 2.1 and deployed to Azure Linux Web Apps Service. The database used is an Azure SQL database. The configuration options include the PushValidator application id, the PushValidator symmetric key, and PushValidator URI for submitting two-factor requests, the PushValidator URI for checking the result of two-factor requests, the database connection string and the logging level. The code is available on GitHub [36].

Duo Demo

This application is the demo target web application which uses Duo Security as its two-factor push notification provider. The application is written using ASP.NET Core 2.1 and deployed to Azure Linux Web Apps Service. The database used is an Azure SQL database. The configuration options include the Duo API hostname, the Duo secret key, the Duo integration key, the database connection string, and the application logging level. The code is available on GitHub [33].

PushValidator iOS App

This application is the iOS app installed on the user's phone which the user must register with the PushValidator Service. The PushValidator app receives the two-factor push notification, scans the QR code from the PushValidator Browser Extension, and sends the authentication response and data points from the QR code to the PushValidator Service. The application is written using Swift 5.0 and installed on an iPhone 6S running iOS version 12.3. The code is available on GitHub [34].

PushValidator Browser Extension

This application is the browser extension that must be installed on the browser the user uses to login to the target web application. The extension collects the IP, URI, and certificate fingerprint of the site the user is authenticating to and generates a QR code containing the data for the PushValidator iOS app to scan.

The extension is written for the Firefox web browser. No other browsers currently support the ability to acquire a site's certificate fingerprint. The extension works in the TOR web browser because it's a customized version of Firefox. The code is available on GitHub [35].

VPN & VPN Client

The VPN service used during testing was the Vypr VPN service [38]. The Vypr VPN client was installed on an Apple MacBook Pro running macOS v10.15.2.

Tor Browser

The Tor browser was downloaded from the official Tor website [39]. The Tor browser was installed on an Apple MacBook Pro running macOS v10.15.2.

Data Flow Authentication and Integrity

Each request and response must be properly authenticated and protected against modification between the target application, the two-factor authentication provider, and the user's mobile device. To ensure that is the case modern cryptographic techniques are used in this implementation. To implement these techniques properly the registration process of the mobile device and the application become paramount in order to securely exchange the relevant cryptographic secrets securely. Many of these techniques are attempting to emulate what existing two-factor authentication providers currently do. The techniques implemented are described below.

Figure 3 show the request and response flow of an authentication attempt in the described implementation. These requests and responses apply cryptographic techniques to each message in order to authenticate and maintain integrity of each request and response. The registration process is used to share the cryptographic secrets between the relevant parties. Request #3 and response #7 use a HMAC of the message to authenticate the sender and guarantee the integrity of the message. This is done by taking each field of the message and concentrating them to serve as the data input to the HMAC and then using a shared secret as the symmetric key which the two-factor provider and the target web application both possess. The symmetric key is generated by the two-factor provider and then given to the web application developer via a web portal who is then responsible for correctly formatting and building their requests. The generation of the payload with the HMAC and verification of the response is aided by a library provided to the application developer. Request #3 and #4 rely on

the security of the push notification service and mobile ecosystem. The security properties of those are out of scope for this implementation

A user must register their mobile device with the two-factor provider through their web portal. The registration process for a device requires a user to login to the two-factor provider web portal where a device Id and a symmetric key is generated. These are shared with the device via a QR code which must be scanned using the two-factor provider's mobile app. The two-factor provider's mobile app then generates an asymmetric key pair using the device's secure enclave (if available) and any additional data needed to send it push notifications. In the case of APNS a device token is required. The public key, device token, and device Id are sent back to the two-factor provider via a HTTP request along with a HMAC. The HMAC uses the previously supplied symmetric key and the concatenation of the aforementioned parameters as the data. Once the two-factor provider receives the request and validates it then the device is fully registered.

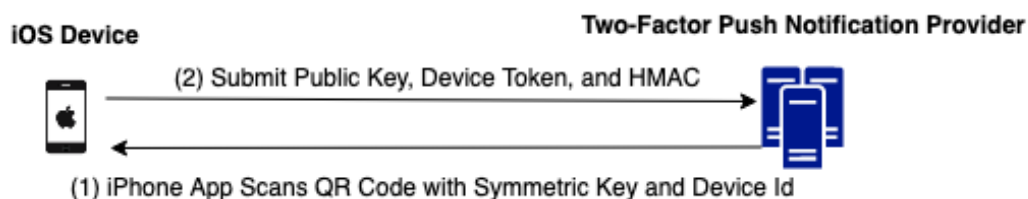


Figure 4 - Device Registration

Response #6 in Figure 3 uses asymmetric cryptography to authenticate and verify the integrity of the message by taking the parameters of the message concatenating them and then signing them with the previously generated private key in its secure enclave. The structure of response #5 can be found in Table 4. The two-factor

provider is able to validate the signature by using the previously provided public key. Once the request is validated it saves the results which the original web application queries in response #7 to determine whether to authenticate the user. The structure of the response in #7 is identical to Table 2 except instead of using a ECDSA signature from the device to verify the authenticity and integrity of the message, a HMAC is generated using the message contents and the application symmetric key. The user is either authenticated or denied via response #8 as shown in Figure 3.

Data Points used to Verify the Client Connection

The implementation gathers the server IP, URI, and certificate fingerprint that the user is actually connecting to. These values are submitted back to the target web application through the two-factor provider so that the target web application can evaluate whether the user is actually connecting to it or to an unknown server which is presumably malicious. The IP is used because typically web applications are available at a consistent set of addresses that the web application owner is aware of. This can often be a set of load balancers, reverse proxies or other connection aggregator infrastructure. This is not always true because of cloud infrastructure though and as a result other data points are needed to help the target web application have enough context to make an appropriate authentication decision.

One of the other data points used is the target web application URI. In most phishing attack scenarios, the URI will be different from the target web applications because the attacker is not able to modify the user's DNS or control the routing of

traffic. The URI, unlike the IP address, is going to be extremely consistent for a web application and unlikely to vary, meaning multiple values would not need to be configured for the check to be successful. There is the possibility that an attacker is able to modify DNS responses so that the legitimate URI points to the phishing site's IP or change the routing of the user's traffic to send requests and responses to the phishing site. In those cases, the URI would match the legitimate sites, but the victim would be visiting a phishing site.

Certificate metadata, specifically the certificate fingerprint, which is a hash of the certificate, can be used to compare the certificate presented by the phishing site against the one the target web application uses. It would be extremely unlikely an attacker is able to get the exact certificate used by the target web application and provide it in such a way the web browser will mark it as valid. The certificate used by a web application for a particular domain will be consistent like the URIs used.

Making these values match in aggregate raises the bar of the attacker's complexity significantly. Additional data points and checks can be used to make an attack even less likely. For example, examining the server IP the user is connecting to against known blacklists, threat intelligence feeds, and Whois data can reveal obvious attackers. Additionally, the target web application can implement logic to compare the server IP data point to authenticating client IPs and if there is a match there is a likelihood that IP is a proxy relaying requests. Such a proxy may or may not be malicious, for example, corporate HTTP proxies could be categorized as non-malicious. Each target

application must weigh the likelihood of false positives and usability that such additional checks and data points will incur.

Unfortunately, non-malicious proxies may be flagged by this implementation as being malicious. The URI would match in such scenarios, but the IP and certificate fingerprint will not. A potential workaround would be to whitelist such proxies' IP and certificate fingerprints. However, this is not ideal and compromises much of the benefit provided by this implementation because a malicious phishing site that resides behind such a whitelisted proxy will be more difficult to detect. In such a scenario the phishing site would more likely to be able to appear as the whitelisted proxies' IP and certificate and the target web application's URI because a LAN environment generally has fewer protections against such attacks. Other non-malicious connection aggregation points such as TOR nodes and VPN servers may also cause this implementation to flag requests. These scenarios are tested with the PushValidator and Duo demo applications under normal circumstances, without an attacker involved.

Browser Extension Role and Mobile Device Alternatives

This implementation requires a browser extension because a trusted piece of software, separate from the JavaScript and HTTP provided by the target web application, must be used to gather the actual data points such as the server IP, URI and certificate fingerprint. The target web application could provide these values but because the MITM would be able to modify what is shown to the user to match the values the server is expecting, the generating JavaScript code from the application

would need to include a HMAC or similar authenticity and integrity check with the data. Such a check would require additional key material to be either passed via the connection which is insecure or would need to have been pre-shared with the user as part of a registration step. This adds a great deal of complexity and the usability of such an implementation is greatly diminished.

As a result, the most convenient option that provides robust integration across many platforms is currently a browser extension. Firefox in particular provides the ability to examine the server certificate and capture the certificate fingerprint. Chrome does not currently provide such an API for its browser extensions. Edge was not tested because at the time of writing it is undergoing a major overhaul to be based on the chromium browser and so is unlikely to provide the API (since Chrome which also based on chromium doesn't) or be stable enough to develop for.

There are downsides to the browser extension such as a more arduous process for the end user because of the additional software that must be installed for every browser they use. The user must also go through the additional step of scanning a QR code after accepting a push notification two-factor request. This additional step is the weakest point in the process as it relies heavily on the user themselves. Potential mistakes include a user scanning a QR code for the site they're not actually authenticating to, the attacker convincing the user to scan the QR code for the legitimate site rather than the phishing site, or the user just not having the technical acumen to complete the process.

In the case where the device the user is requesting authentication from is a mobile device, different approaches are required depending on the mobile browser OS. A browser extension is more difficult on iOS mobile devices because iOS only allows for limited capability browser extensions through its Web Kit API which is what all iOS browsers are based on, not just Safari [40]. There are application extensions, called action and sharing extensions, which allow limited inter-application interaction. Currently, the workaround idea on iOS would be to require the mobile device to open a browser to the site within the two-factor provider mobile app in order to access the relevant data points. The downside to this approach is that for any target web application that uses the described implementation would only be able to be accessed on an iOS mobile device from the two-factor provider's app. Android mobile devices can use browser extensions so the approach would be for the extension to provide a deep link to the two-factor provider mobile app with the data points as parameters [41]. In the reference implementation here an iOS app is developed but an Android app was not. The developed iOS app does not provide a solution for accessing the target web application on the iOS device itself.

Message Fields

The following tables show the fields contained in each message during an authentication request and device registration

Two-Factor Authentication Request

This is the request sent from the target application to the two-factor service provider after the user has successfully entered their username and password. This will trigger the provider to send a push notification to the PushValidator app on the mobile device the authenticating user has previously registered with the two-factor provider.

Field	Type	Description
ApplicationId	String	Web Application's unique identifier used to grab associated application's data such as name and symmetric key.
Secret	String	HMAC generated from a string consisting of all other fields concatenated and the target application's symmetric key
Username	String	Username used to make the authentication request.
ClientIp	IP Address	IP address of the client requesting authentication
Timestamp	DateTime	Timestamp of request

Table 1 - Two-Factor Authentication Request from Target Application to PushValidator Service

The **TransactionId** is returned in response to this request by the PushValidator service to the target application which allows the target application to subsequently query the result of the authentication request.

Two-Factor Authentication Result

Retrieved from the Two-Factor provider by the web application periodically polling the provider endpoint using the transaction id as a lookup key. The two-factor provider is able to provide the result once the user has sent a response from the Push Validator app on their mobile device.

Field	Type	Description
TransactionId	String	Unique value for tracking the request
Result	String	Success or Failure
ServerIp	String	IP of the server the client is authenticating to
CertificateFingerprint	String	Fingerprint of the certificate the client is authenticating to
ServerURI	String	URI of the server the client is authenticating to
Signature	String	HMAC generated from a string consisting of all other fields concatenated and the target application's symmetric key

Table 2 - Two-Factor Authentication Result returned from PushValidator Service to Target Application

Push Notification

Sent by the two-factor service provider to the PushValidator app on the authenticating user's mobile device. This triggers the user to accept or decline the authentication request and then to scan the QR code generated by the PushValidator browser extension which contains the authentication data points to be sent back to the two-factor service provider and ultimately the target web application.

Field	Type	Description
Username	String	Username used to make the authentication request.
ApplicationName	String	Web Application name user is attempting to authenticate to that is registered with the PushValidator service.
GeoLocation**	String	Geolocation of the ClientIp based on external GeoIP databases.
ClientIp	String	IP address of client requesting authentication
TransactionId	String	Unique value for tracking the request
Timestamp	DateTime	Timestamp of request

Table 3 - Content of Push Notification message sent from PushValidator Service to the authenticating user's mobile device

**** Not currently implemented**

Push Notification Response

Response sent by the PushValidator app to the PushValidator service after the user has denied the request or accepted it and scanned the QR code generated by the PushValidator browser extension. This message is subsequently relayed to the target application with a HMAC of the contents generated using the target application's symmetric key rather than the device's ECDSA signature.

Field	Type	Description
UserId	String	Unique user identifier
TransactionId	String	Unique value for tracking the request
ClientIp**	IP Address	User's actual IP address of authenticating client.
ServerURL*	String	User's actual URL they're authenticating to
ServerIP*	IP Address	Actual IP address hosting the application the user is attempting to authenticate to.
CertificateFingerprint*	String	Actual fingerprint of certificate presented by the application the user is attempting to authenticate to.
Signature	String	Cryptographically signed concatenated string of all other fields in request using the private key stored securely on the user's device.

Table 4 - Response to the Push Notification sent from the PushValidator iOS App to the PushValidator Service

*** Obtained from the PushValidator browser extension.**

**** Not currently implemented**

Device Registration

Sent by the device during the registration process after scanning the QR code generated by the two-factor provider which contains the device id and a symmetric key.

The symmetric key is used to generate a HMAC of the response data to validate this message is from the expected device.

Field	Type	Description
DeviceId	String	Id of device provided by the QR code during initial registration
Secret	String	HMAC generated by a string of all other fields in this message concatenated and the symmetric key provided by the QR code during registration
PublicKey	String	Public Key of the corresponding private key generated by the Push Validator app on the user's mobile device that is being registered. The public key will be used to validate future push notification responses that will have contain a signed data string.
DeviceToken	String	Token needed by the Push Validator two-factor provider service to send push notifications through Apple's push notification service.

Table 5 - Registration message sent from the PushValidator iOS App to the PushValidator Service

Additional Benefits

Aside from the additional authentication validation provided by the described implementation, there are additional security benefits that can be derived. One such is immediate threat intelligence data about phishing sites that target the web application's users. When an authentication attempt is deemed to be potentially malicious, the server IP, URI and certificate fingerprint collected from the attempt can be stored and tracked by the web application or two-factor provider. The benefit to the web application is more siloed as it will only be able to see the malicious applications targeting its users, but this provides the ability to prevent connections from ever starting by blacklisting and denying match IP addresses or alerting their entire user base to such sites. The two-factor provider stands to gain even more benefit because they'll be able to aggregate such data across all applications and users. This in turn allows them to similarly build blacklists of the server IPs, URIs and certificates of such malicious phishing applications. The provider can use these blacklists to deny requests users mistakenly accept or ones that the target web application may incorrectly accept (since the acceptance logic is ultimately up to the target web application) across all of their customers. So, if several users from customer A has their login attempt flagged which all match a particular set of data points then the two-factor provider can proactively block matching requests from users of other customers of their service.

Pitfalls

Beyond the issues with incorrectly flagging legitimate proxies discussed above, there are other downsides to this implementation which may limit its usability. First, while having to have a separate mobile application for the two-factor provider is somewhat of an inconvenience, it is commonplace for most two-factor providers and is definitively the case with all push notification two-factor providers because of how mobile push notifications work. The extra piece of trusted software, in this implementation the browser extension, is an additional hassle. The browser extension is easy to install and non-invasive but in certain high security or corporate environments there may be significant limitations on installing such extensions. Additionally, it is another piece of software that must be maintained and secured and thus additional attack surface for an attacker to target. It is a crucial step to the implementation, so it is likely to be heavily focused on by attackers. The flip side is that the browser extension code and purpose is very simple because it simply provides a QR code with some basic details of the site the user is currently on. Additionally, the browser extension ecosystem has been much maligned over the past few years with no end in sight due to malicious copies of existing extensions being easy to develop and publish to the extension marketplaces. There are few checks in place to prevent malicious replicas of an extensions so if an attacker was able to convince a user to install one in place of the trusted one from the two-factor provider they could circumvent the protections provided. There is the potential for cryptographic techniques to be added to ensure the data provided by the extension to the two-factor provider mobile app is in fact from the

correct extension, but these are likely to add more cumbersome registration and installation requirements.

Having the target web application ultimately perform the logic to authenticate the user based on the response data points provided by the mobile application to the two-factor provider is an opportunity for issues to arise as well. Namely the target web application could have bugs in their implementation that incorrectly regard the data points provided as valid when they should not be. The web application developer may also be unable to gather the expected values for the data points or otherwise unable to dynamically change them at runtime which results in a high rate of false positives. This case is more likely for the server IP data points as discussed previously. Having the two-factor provider perform such logic would ease this burden but would require the target web application developer to provide the expected data point values such as the server IPs, URIs and certificate fingerprints to the two-factor provider. Another solution could be for the two-factor provider to provide SDKs that contain pre-built code for interacting with the two-factor provider service from the target web application code that require minimal development effort. This is in fact what Duo Security does [29] and how the sample attack was created. The additional downside with the SDK approach is that the two-factor provider must provide several different SDKs, one for each programming language and web application framework.

One of the biggest issues with this implementation is that if an attempt is flagged it means the user has already provided their username and password to the malicious phishing site and thus those credentials are compromised. This presents a clear issue

whereby the target web application should no longer accept those credentials and if the user has used those same credentials at other sites, they should reset them there too. There is no clear workaround to this issue other than inverting the order of when credentials are provided. In such a scenario the web application would require the user to enter their username first which would cause the push notification authentication process to fire first, once that process is successful then the target web application would prompt the user for their password as the second factor. This prevents the malicious phishing site from obtaining the user's password in scenarios where the described implementation flags the authentication attempt as potentially malicious.

This implementation relies on the security of additional third parties, namely the push notification service and the two-factor authentication providers. This is not uncommon in today's modern web application and authentication landscape but should be considered in web application developers threat modeling exercises. Because these providers are often closed source and do not allow open external penetration testing or audits, they are essentially black boxes which users and developers must trust to provide such services securely. This is obviously not ideal for high security environments. The cryptographic techniques used in the described implementation do attempt to mitigate any security issues with the push notification services by signing and verifying the responses from the devices with asymmetric keys. However, the use of symmetric keys and holding device responses diminishes the overall security posture by making the two-factor authentication provider a potential weak point in the process. A way to mitigate the issue further would for the target web application to obtain the

user's device public key directly from the user and then provide it to the two-factor authentication provider. The two-factor provider could then provide the original response message from the device to the target web application who could independently verify the response.

V. Results

The following scenarios test Duo and PushValidator two-factor services. For the attack scenario the setup described on page 18 s is used. The other components are described under the Implementation Components section.

Testing Modlishka Against an Application using Duo Two-Factor Push Notifications

The target application in this scenario used Duo Security as its two-factor push notification provider. The target application was a scaffolded ASP.NET Core 2.1 web app using the built in ASP.NET Core Identity framework for authentication. Minor modifications were made to the scaffolded application to integrate Duo. Those changes include adding two controller methods, one to create the authentication request for the Duo service which is passed to the Duo provided JavaScript and another to verify the response from the Duo service. The generation of the Duo request and verification of the subsequent response were handled by Duo's .NET SDK library. Minor build related modifications were made to the library because as of the time of writing it was target only .NET 3.5 which is not compatible with .NET Core. The target web application was run on Azure's Linux Web App service under the Basic tier. Duo's iOS mobile app was installed on an iPhone 6S device using iOS v12.3. An image of the login page of the Duo demo application with the normal URI is show in Figure 5.

Log in

Use a local account to log in.

Email

Password

Remember me?

Log in

[Forgot your password?](#)

[Register as a new user?](#)

Use another service to log in.

There are no external authentication services configured. See [this article](#) for details on setting up this ASPNET application to support logging in via external services.

© 2019 - DuoDemo

Figure 5 - Duo Demo Application

There was an initial registration step whereby the user was created within the target web application where two-factor authentication had to be enabled. An initial login attempt was conducted to register the user's device with the Duo service. The attack scenario assumes those steps had already been completed prior.

Figure 6 shows the initial login page for the target web application with the key indicator the attack is ongoing being the URI. It should be noted the browser is showing no security warnings because the certificate is considered valid.

Log in

Use a local account to log in.

Email

Password

Remember me?

Log in

[Forgot your password?](#)

[Register as a new user?](#)

Use another service to log in.

There are no external authentication services configured. See [this article](#) for details on setting up this ASP.NET application to support logging in via external services.

© 2019 - DuoDemo

<https://pushvalidator.com/Home/Contact>

Figure 6 - Duo Demo Application Login with Attack in Progress

Figure 7 shows the demo application's two-factor page where the Duo JavaScript submits the authentication request and waits on a response.

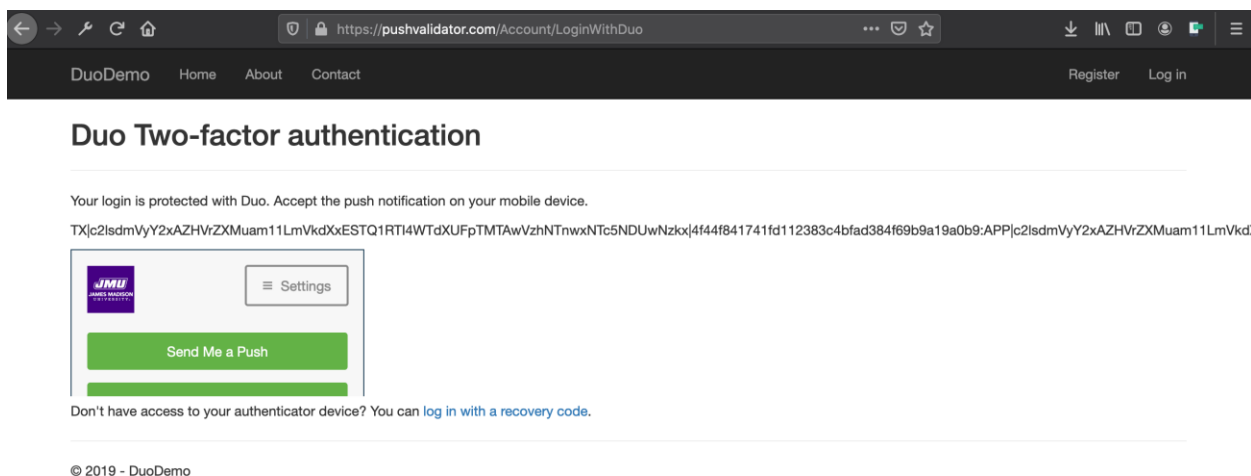


Figure 7 - Duo Demo Application Push Two-Factor Page with Attack in Progress

Figure 9 shows the push notification received on the user's mobile device. Interestingly the IP address of the user shown in Figure 9 is in fact correct and not of the attacker's application IP. The attacker's logs in Figure 8 from the Modlishka application show that the man-in-the-middle real-time relay attack is successful and the `AspNetCore.Identity.Application` cookie is obtained. With this cookie the attacker is able to impersonate the user and hijack their session.


```

COOKIES
=====
Timestamp: Sunday, 19-Jan-20 16:16:14 UTC
=====
RemoteIP: 174.53.73.221:49596
=====
UUID: CfdJ80Mlme10tBBEnsHH8CQdkf7PNMSpbfdKlajQVlnevGD_gpNFbtXJ4zZ1VDTXLq_-Rf491m2w5oXkC6vc8AJUCR6sP2zHbeazg54p00TTYJ
=====
URL: https://duodemo.azurewebsites.net
=====
Identity.TwoFactorUserId=; expires=Thu, 01 Jan 1970 00:00:00 GMT; path=/; samesite=lax###.AspNetCore.Identity.Appli
=====

REQUEST
=====
Timestamp: Sunday, 19-Jan-20 16:16:14 UTC
=====
RemoteIP: 174.53.73.221:49596
=====
UUID: CfdJ80Mlme10tBBEnsHH8CQdkf7PNMSpbfdKlajQVlnevGD_gpNFbtXJ4zZ1VDTXLq_-Rf491m2w5oXkC6vc8AJUCR6sP2zHbeazg54p00TTYJ
=====
GET / HTTP/1.1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, br
Accept-Language: en-US,en;q=0.5
Connection: keep-alive
Cookie: .AspNetCore.Identity.Application=CfdJ80Mlme10tBBEnsHH8CQdkf5HpJXcxg1x06wK2uuBLcj1CQwBFPLmkg10KQV32RzHYoagTYh
Referer: https://duodemo.azurewebsites.net/Account/LoginWithDuo
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:73.0) Gecko/20100101 Firefox/73.0

```

Figure 8 - Modlishka MITM Duo Attack Log

The first key take away is that the Duo service is seemingly unaware or otherwise does nothing about the connection having a man in the middle. The attacker is able to collect the user's session cookie after the two-factor push notification is accepted and the target web application receives the successful response from Duo's JavaScript. To make matters worse the client IP address shown in the push notification is that of the user's browser which seems to have been collected by the Duo service itself from the request made from the Duo JavaScript. This means the user has absolutely no indication from the two-factor provider that there is a man in the middle since the user's IP and related geolocation are correct and are the key data points a user is expected to use to identify a malicious login attempt other than timing.

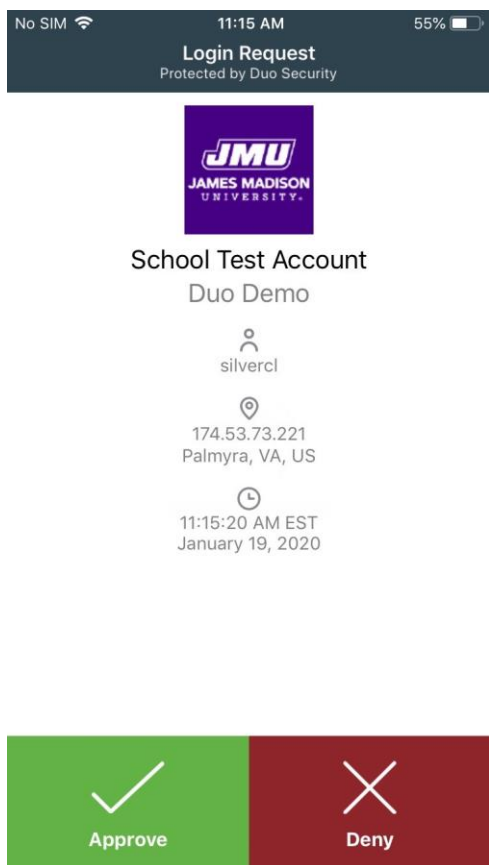
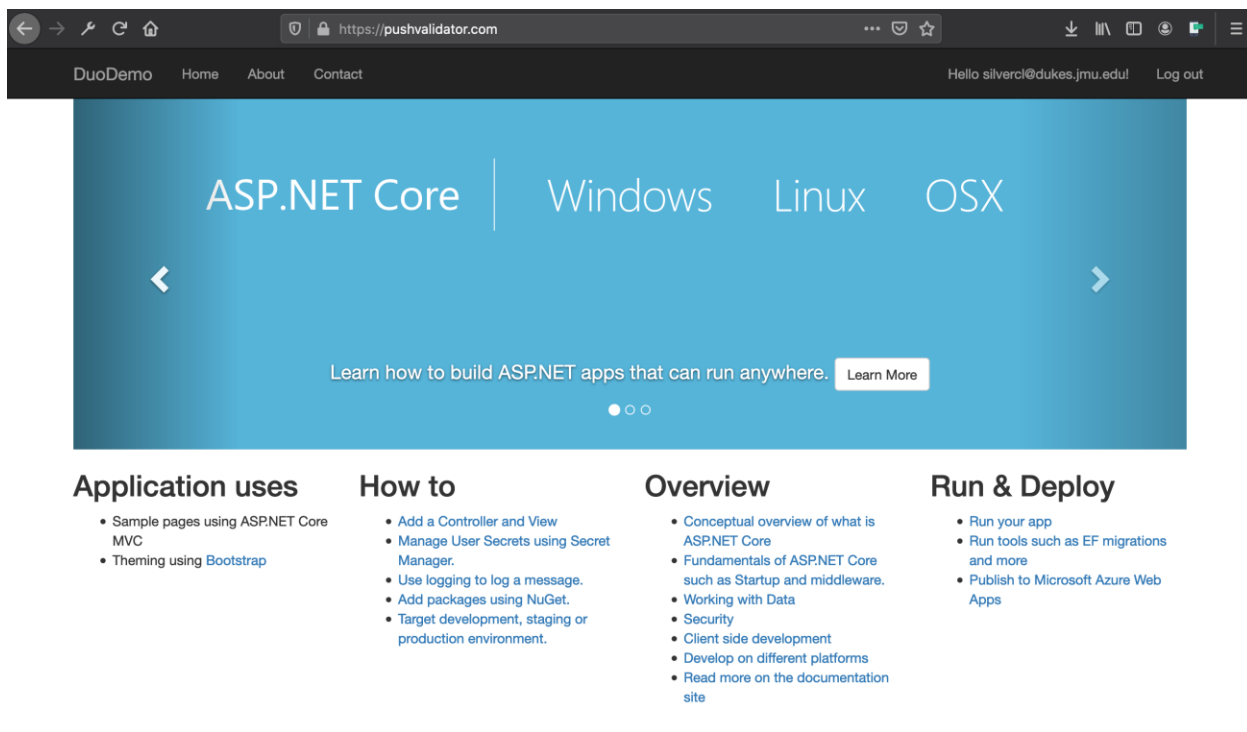


Figure 9 - Duo iOS App Push Notification with Attack in Progress



<https://pushvalidator.com/#myCarousel>

Figure 10 - Duo Demo Application Successful Login with Attack in Progress

Setup of PushValidator Service

A user who wishes to authenticate to an application using the PushValidator service must register their device with the PushValidator service via a web application interface.

Figure 11 shows the page a user sees when registering their device. Their presented a QR code which contains their device ID and a symmetric key. The user then selects the register button on the PushValidator iOS app which prompts the user to scan the QR code. Once the QR code is scanned the device generates a public/private key pair in its secure enclave. The PushValidator app then takes the public key, the device ID, the device token used to send push notifications via the APNS, and a HMAC of the

aforementioned fields using the symmetric key from the QR code and submits that to the PushValidator service to complete registration. The message is shown in Table 5.

The screenshot shows a web browser window with the URL `https://pushvalidatorservice.azurewebsites.net/Devices/Details/91bfe943-5840-452e-976f-f4de67fe96d1`. The page title is "Device Details". Below the title is a table of device information:

Id	91bfe943-5840-452e-976f-f4de67fe96d1
Name	iPhone 6S
SymmetricKey	sQtX
Registered	<input checked="" type="checkbox"/> 3BqJgU=
PublicKey	MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAELEYePbozjNFO7Z7QH7BLOydjHB6Ga2o73JQQa7jk3EMk07OcorY6uHz2nnGWllom08SNgtleYqsZXPPoZecQ==
DeviceToken	e75z f56a43

Below the table, there is a QR code and the text: "Scan the QR Code to attempt to register your device." Below the QR code are two links: "Edit" and "Back to List". At the bottom of the page, there is a copyright notice: "© 2020 - PushValidator".

Figure 11 - Registering a device in the PushValidator service

An application developer must register their application with the PushValidator service in order to use it as a two-factor provider. To do this the developer logs in to the PushValidator service and receives an application ID and a symmetric key which they must use with the PushValidator SDK for their web application. Figure 12 shows the page a developer sees when registering their application with the PushValidator service.

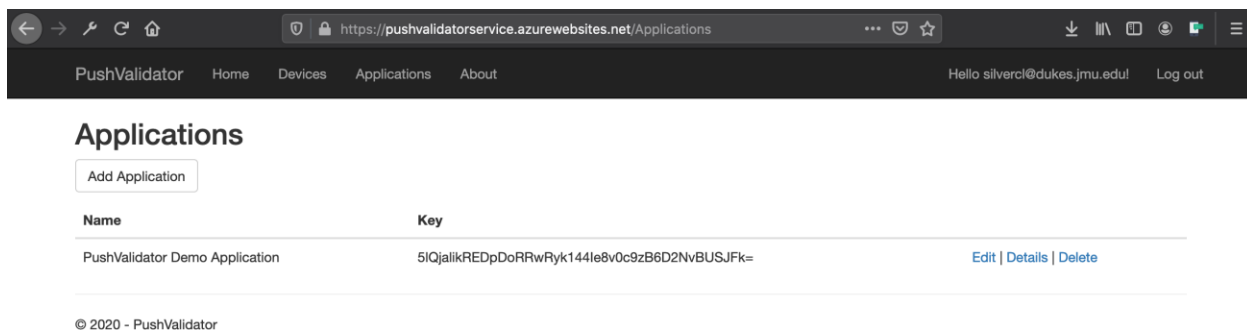


Figure 12 - Application registration in the PushValidator service

Testing Modlishka Against Application using PushValidator

The target web application in this scenario uses the described PushValidator implementation as its two-factor push notification provider. The target application was a scaffolded ASP.NET Core 2.1 web app using the built in ASP.NET Core Identity framework for authentication. Minor modifications were made to the scaffolded application to integrate PushValidator. Those changes include adding two controller methods, one to create the authentication request for the PushValidator service which is passed to the PushValidator provided JavaScript and another to verify the response from the PushValidator service. The generation of the PushValidator request and verification of the subsequent response were handled by PushValidator's .NET SDK library. The target web application was run on Azure's Linux Web App service under the

Basic tier. PushValidator's iOS mobile app was installed on an iPhone 6S device using iOS v12.3. An image of the login page of the Duo demo application with the normal URI is show in Figure 13.

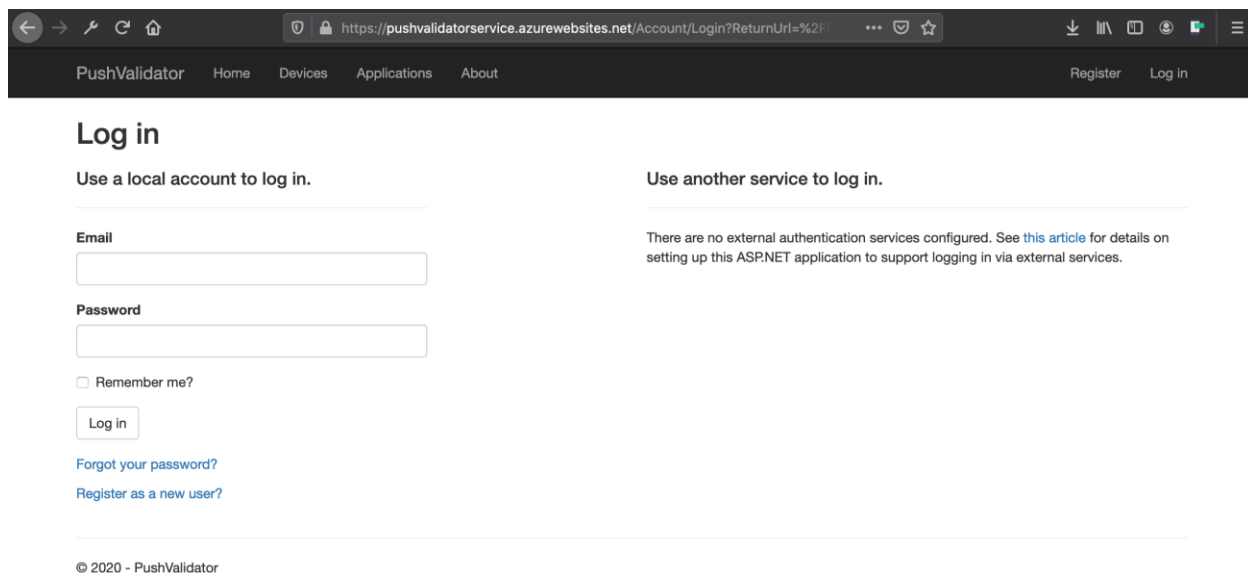


Figure 13 - PushValidator Demo Application

There was an initial registration step whereby the user was created within the target web application where two-factor authentication had to be enabled. The user also had to login to register their device with the PushValidator service. The attack scenario assumes those steps had already been completed prior.

Figure 15 shows the push notification received from the PushValidator service. The IP of the user shown is that of the attacker because the target application is responsible for collecting the value which serves as an indicator that something is not

correct. After clicking on the check mark the user will be prompted to scan a QR code generated by the PushValidator browser extension.

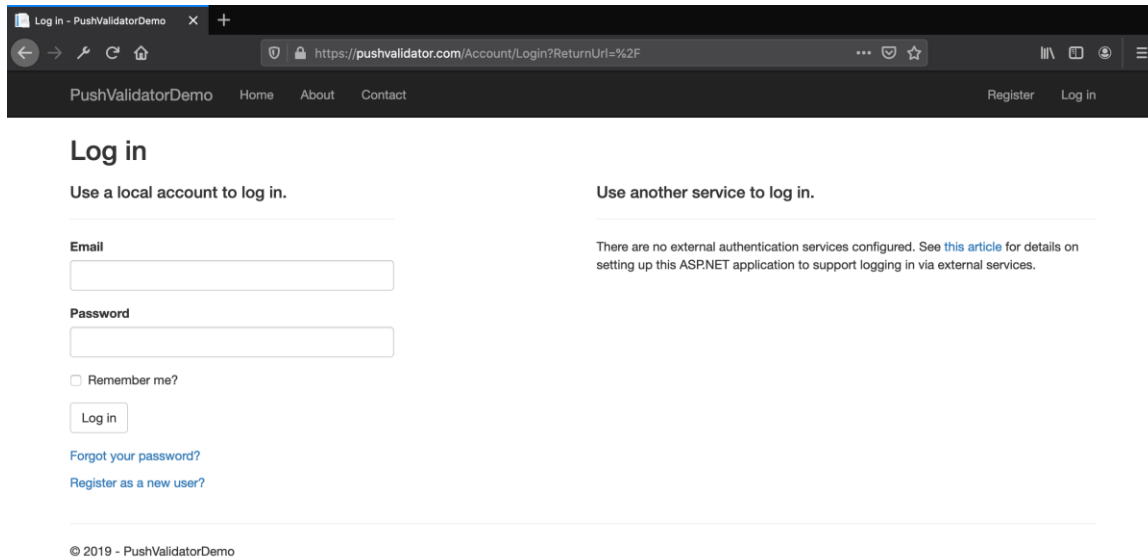




Figure 14 - PushValidator Demo Application with Attack in Progress

No SIM  11:50 AM 54% 

Application: PushValidator Demo Applicat

Client IP: 40.123.27.244

User ID: silvercl@dukes.jmu.edu

Transaction ID: e7857651-48cd-46bd-8928-b475da11f821

Timestamp: 2020-01-19 11:49:56

Geo Location: Not yet implemented



Figure 15 - PushValidator iOS Push Notification

The QR code is shown in Figure 16 and has the server IP, URI and certificate fingerprint of the attacker's application. Those values are sent back to the target web application via the PushValidator two-factor service from the PushValidator iOS app.

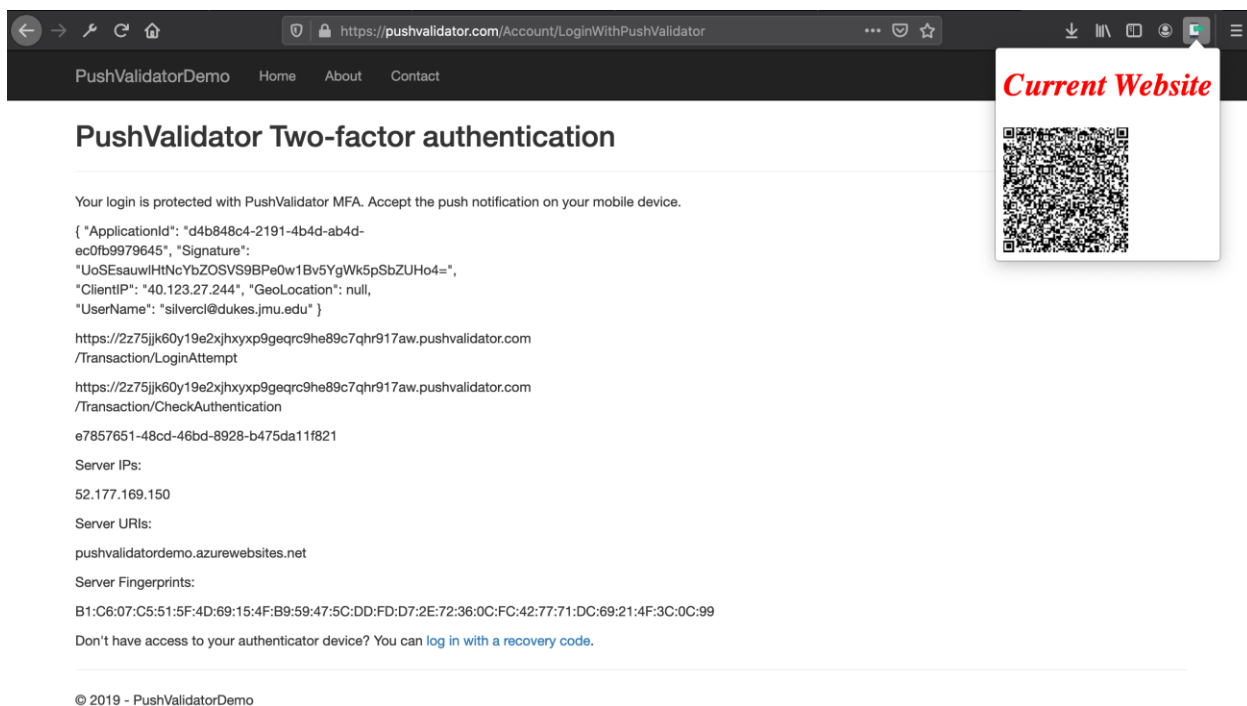


Figure 16 - PushValidator Demo Application Two-Factor Push Notification Page with Attack in Progress

Included are images showing the user authenticating to the target web application, the PushValidator JavaScript submitting the authentication request and waiting on a response, the push notification received on the user's mobile device and the attacker's logs from the Modlishka application.

The target web application is able to identify that the connection has an active MITM because the server IP, URI and certificate fingerprint do not match the application's known respective values. The application then denies the attempt and the attacker is never able to obtain a valid session cookie. An interesting implementation difference between Duo and the described PushValidator implementation is that the target web application is responsible for collecting the authenticating client's IP address

in the PushValidator implementation. This results in the user being able to see the attacker's application IP address as the authenticating IP in the push notification. This gives the user an increased chance of catching such a MITM attack compared to Duo's implementation.

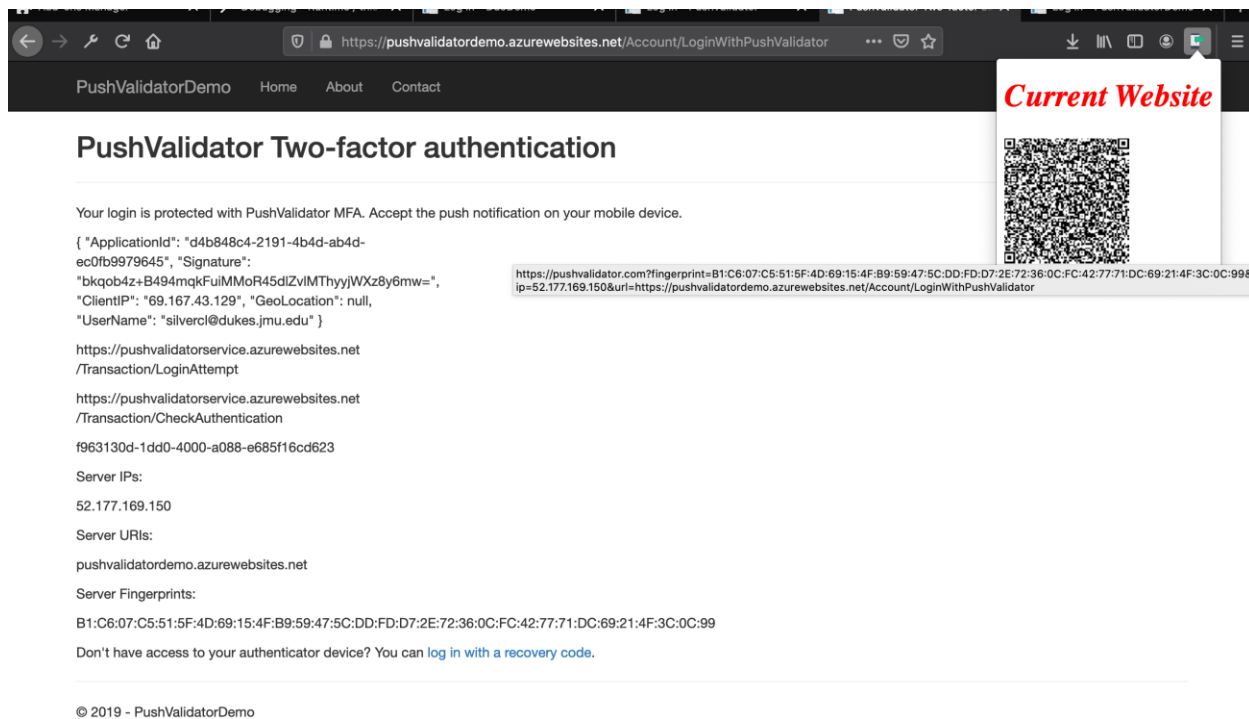
It's likely Duo made their choice because having the application developer be responsible for collecting the user's IP is likely to be too complex for the average developer and thus an unreliable piece of data. As an example of the complexity of collecting the user's IP accurately, during the implementation of PushValidator the client's IP was originally collected via the *RemoteIpAddress* property on the `HttpContext.Connection` object provided by the ASP.NET Core SDK, however when deploying the application to Azure this value was incorrect because Azure performs its own reverse proxying of the connection in order to perform its own TLS termination so the implementation instead had to collect the IP from a HTTP header provided by the Azure reverse proxy, `HttpContext.Request.Headers["X-Client-IP"]`. It's important to note that the reverse proxy provided by Azure is a black box and most developers would be completely unaware it existed. PushValidator performs as expected and catches the real-time relay MITM attack provided by Modlishka in this scenario.

Testing PushValidator with a Client using a VPN

The target web application, user, and mobile device are the same as in the above scenario except the user is connecting to the target web application through a VPN

connection. The VPN connection is a commercial VPN service where all network traffic is tunneled through it. There is no attacker in this scenario.

The user logs in as previously described into the PushValidator demo application using their username and password. They then see the page shown in Figure 17 which triggers the push notification seen in Figure 18.



PushValidatorDemo Home About Contact

Current Website

PushValidator Two-factor authentication

Your login is protected with PushValidator MFA. Accept the push notification on your mobile device.

```
{ "ApplicationId": "d4b848c4-2191-4b4d-ab4d-ec0fb9979645", "Signature": "bkqob4z+B494mqkFuiMMoR45dIZvIMThyyjWXz8y6mw=", "ClientIP": "69.167.43.129", "GeoLocation": null, "UserName": "silvercl@dukes.jmu.edu" }
```

https://pushvalidatorservice.azurewebsites.net/Transaction/LoginAttempt

https://pushvalidatorservice.azurewebsites.net/Transaction/CheckAuthentication

f963130d-1dd0-4000-a088-e685f16cd623

Server IPs:

52.177.169.150

Server URIs:

pushvalidatordemo.azurewebsites.net

Server Fingerprints:


B1:C6:07:C5:51:5F:4D:69:15:4F:B9:59:47:5C:DD:FD:D7:2E:72:36:0C:FC:42:77:71:DC:69:21:4F:3C:0C:99

Don't have access to your authenticator device? You can [log in with a recovery code](#).

© 2019 - PushValidatorDemo

https://pushvalidator.com?fingerprint=B1:C6:07:C5:51:5F:4D:69:15:4F:B9:59:47:5C:DD:FD:D7:2E:72:36:0C:FC:42:77:71:DC:69:21:4F:3C:0C:99&ip=52.177.169.150&url=https://pushvalidatordemo.azurewebsites.net/Account/LoginWithPushValidator

Figure 17 - PushValidator Demo Application Accessed through a VPN

No SIM  12:57 PM 54% 

Application: PushValidator Demo Applicat

Client IP: 69.167.43.129

User ID: silvercl@dukes.jmu.edu

Transaction ID: f963130d-1dd0-4000-a088-e685f16cd623

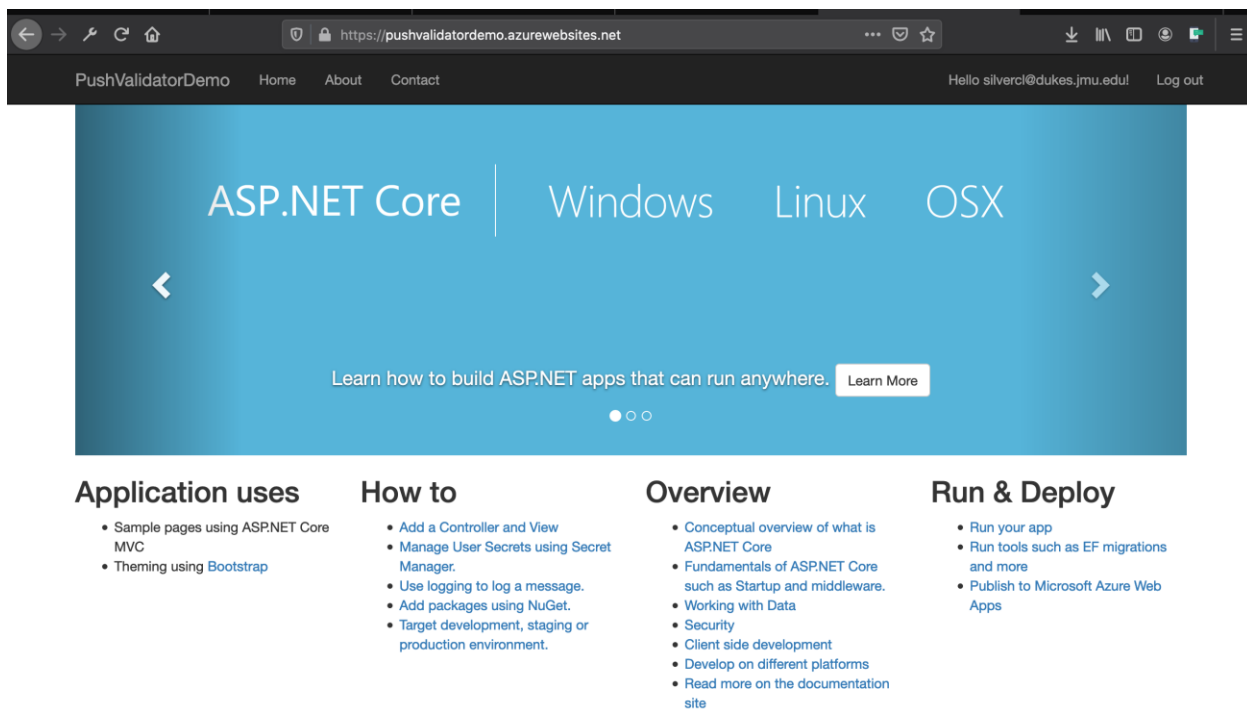
Timestamp: 2020-01-19 12:57:31

Geo Location: Not yet implemented



Figure 18 - PushValidator iOS App Push Notification Initiated by User Login through a VPN

The client IP shown in Figure 18 is that of the VPN connections exit point and not that of the user's actual public IP. The QR code shown in Figure 17 has the correct values for the PushValidator demo application and the login is successful after the user accepts the notification in Figure 18 and scans the QR code when prompted. Figure 19 shows the successful login result.



<https://pushvalidatordemo.azurewebsites.net/#myCarousel>

Figure 19 - PushValidator Demo Application Successful Login with User Accessing through a VPN

The PushValidator implementation is shown to work with a VPN service and behaves the exact same as Duo does.

Testing PushValidator with a Client using TOR Browser

The target web application, user, and mobile device are the same as in the above scenario except the user is connecting to the target web application through the TOR network. The connection is established using a special browser, specifically the official TOR browser, which is a derivative of the Firefox browser modified to route its traffic through the TOR network.

Figure 20 shows the PushValidator demo application two-factor login page accessed through the Tor browser. The page triggers the push notification to be sent to

the PushValidator iOS app. The push notification is shown in Figure 21. Upon accepting the push notification, the user is prompted to scan a QR code shown in Figure 22. The values presented by the QR code are shown more clearly in Figure 23. The server IP is notably 0.0.0.0 which is unsurprising given how Tor routes its traffic. The value is likely being obfuscated or is otherwise unavailable as a result.

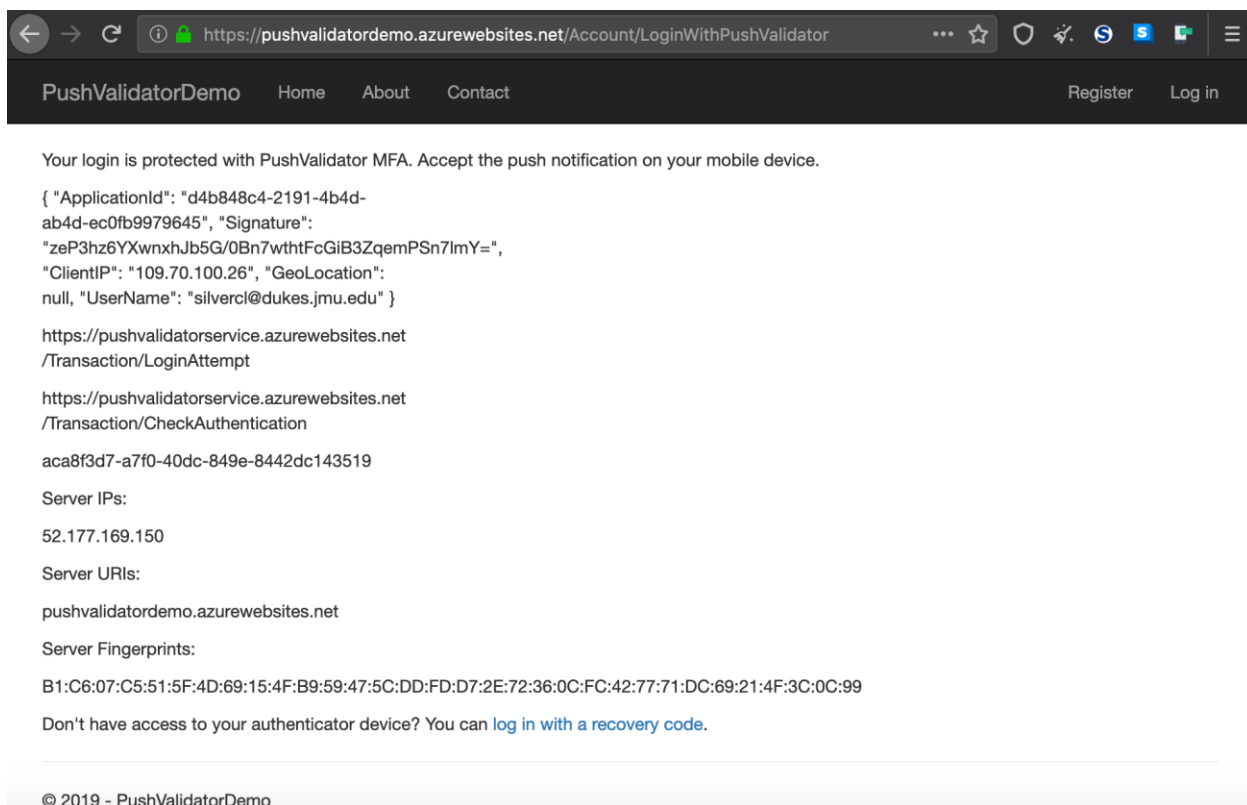


Figure 20 - PushValidator Demo Application Two-Factor Page Accessed through the TOR browser

No SIM 12:27 PM 54%

Application: PushValidator Demo Applicat
Client IP: 109.70.100.26
User ID: silvercl@dukes.jmu.edu
Transaction ID:aca8f3d7-a7f0-40dc-849e-8442dc143519
Timestamp: 2020-01-19 12:27:06
Geo Location: Not yet implemented



Figure 21 - PushValidator iOS App Push Notification Initiated by user Logging in with the TOR browser

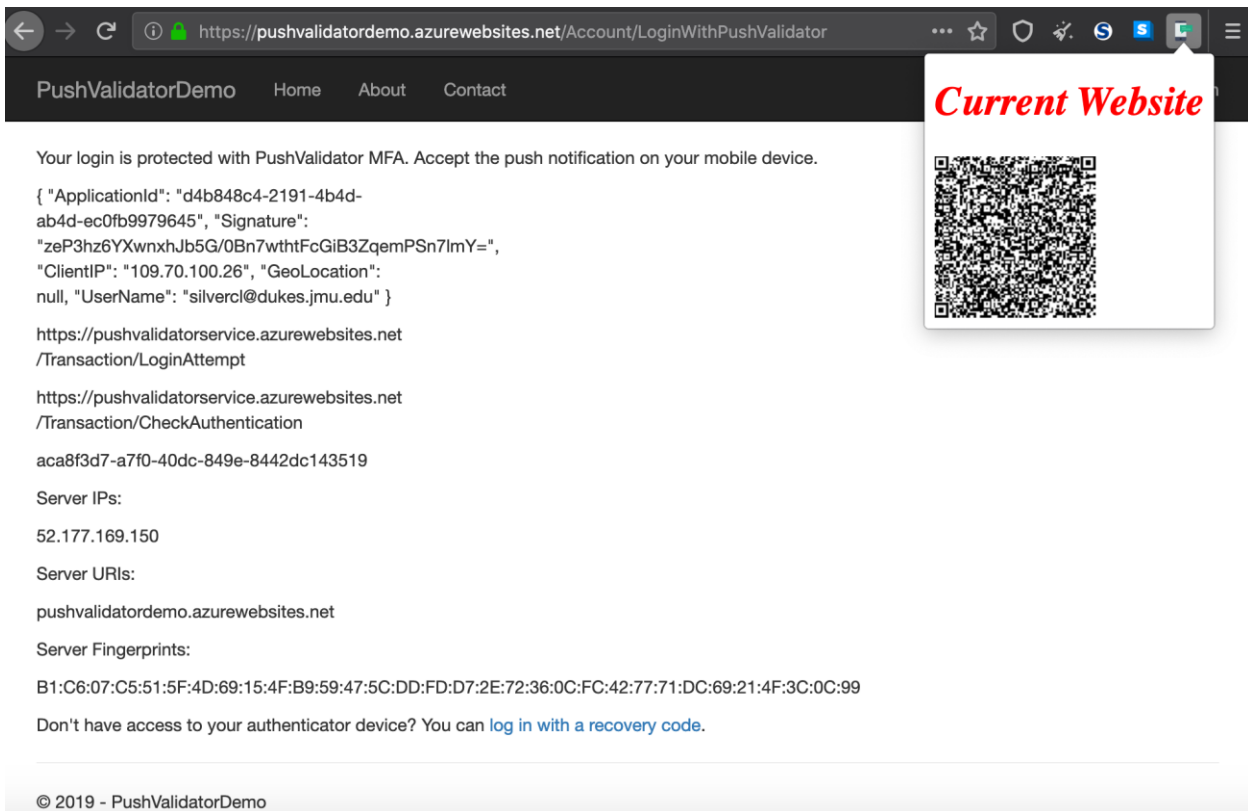


Figure 22 – QR Code that must be scanned by PushValidator iOS App

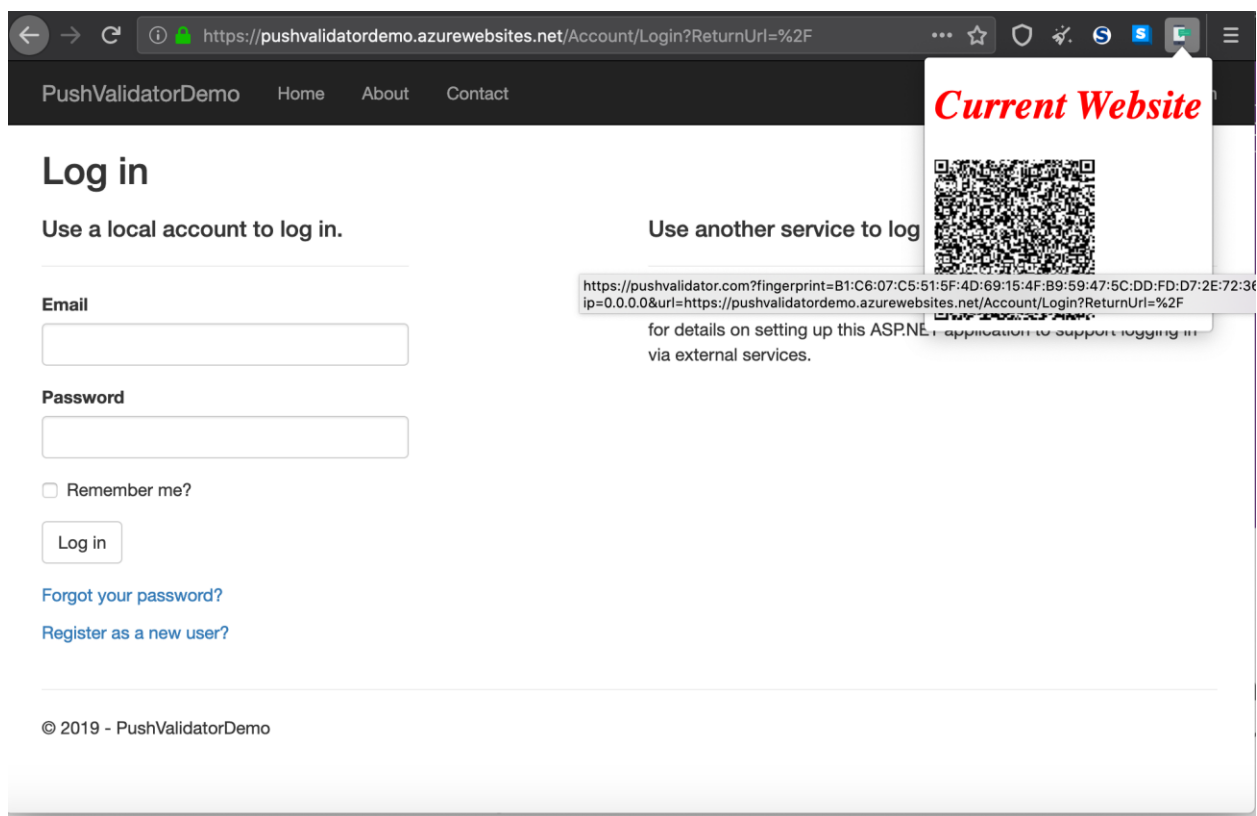


Figure 23 - Caption showing parsed PushValidator QR code data

The server IP being 0.0.0.0 results in the PushValidator application rejecting the login since it doesn't match its IP. As is, PushValidator will not work when accessing an application via the Tor browser.

Testing Duo with a Client using a VPN

The target web application, user, attacker and mobile device are the same as in the above scenario except the user is connecting to the target web application through a VPN connection. The VPN connection is a commercial VPN service where all network traffic is tunneled through the VPN connection.

Figure 24 shows the user via a VPN connection accessing the Duo demo application two-factor login page which triggers a push notification to be sent to the user's mobile device. Figure 25 shows the subsequent push notification the user receives. The IP address of the client is the server the user's connection exits through and the IP is geolocated to Ashburn, VA. This geolocation is less than 150 miles from the actual IP address of the client.

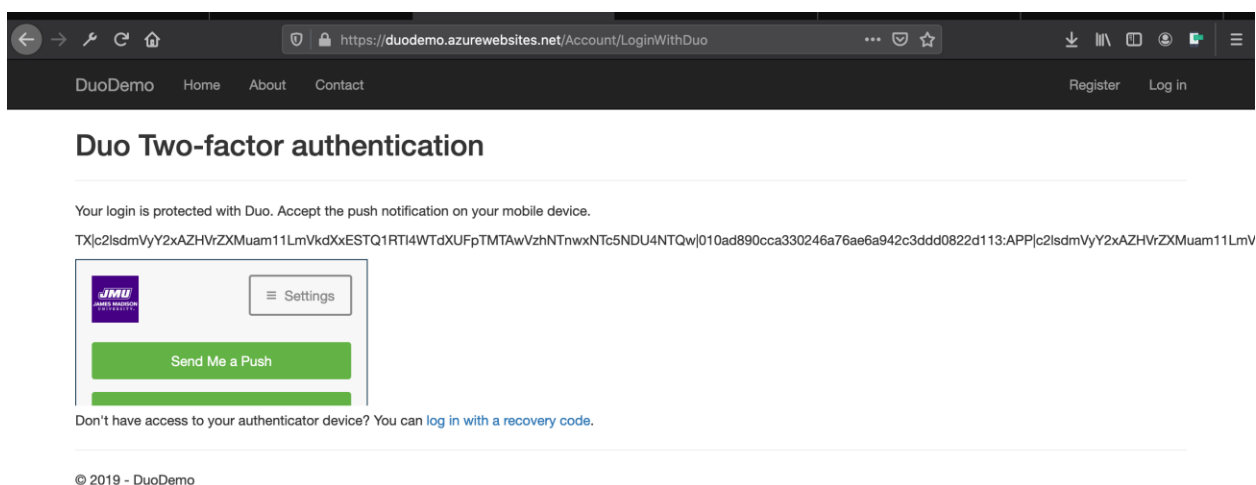


Figure 24 - Duo Demo Application Two-Factor Page Accessed through a VPN

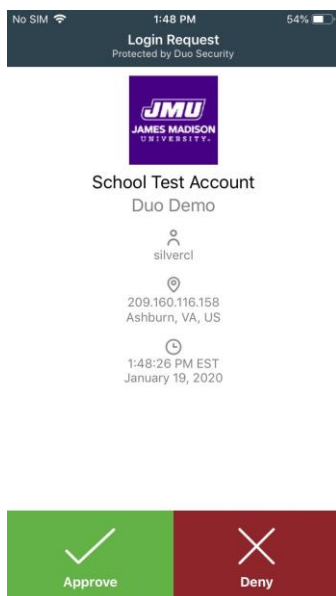


Figure 25 - Duo iOS App Push Notification Initiated by User Logging in through a VPN

The user is able to successfully authenticate to the application. It should be noted Duo does provide additional settings to prevent logins from anonymous networks such as TOR and VPN services.

Testing Duo with a Client using TOR Browser

The target web application, user, and mobile device are the same as in the above scenario except the user is connecting to the target web application through the TOR network. The connection is established using a special browser, specifically the official TOR browser, which is a derivative of the Firefox browser modified to route its traffic through the TOR network.

Figure 26 shows the user logging in via the Tor browser to the Duo demo application and being shown the Duo two-factor page. The page initiates a push

notification to the Duo app on the user’s mobile device which is shown in Figure 27. The IP address notably has no geolocation tied to it and as a result is shown as unknown.

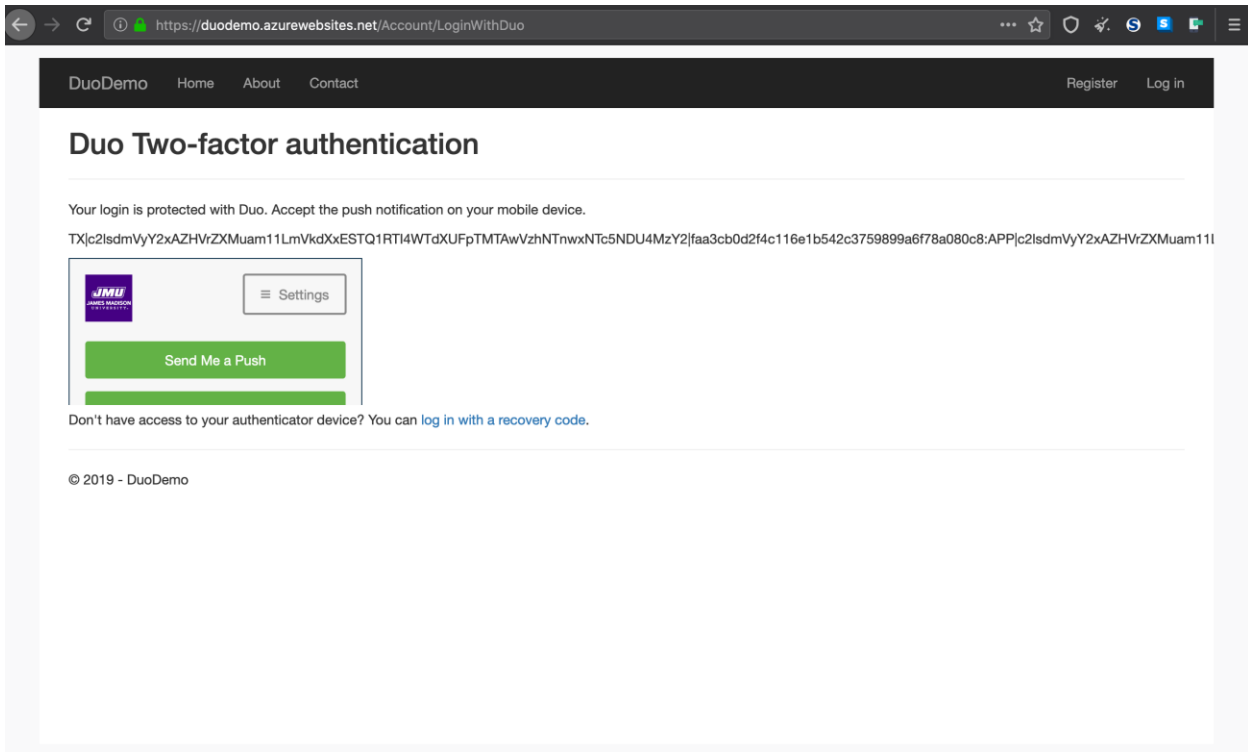


Figure 26 - Duo Demo Application Two-Factor Page Accessed from the Tor browser

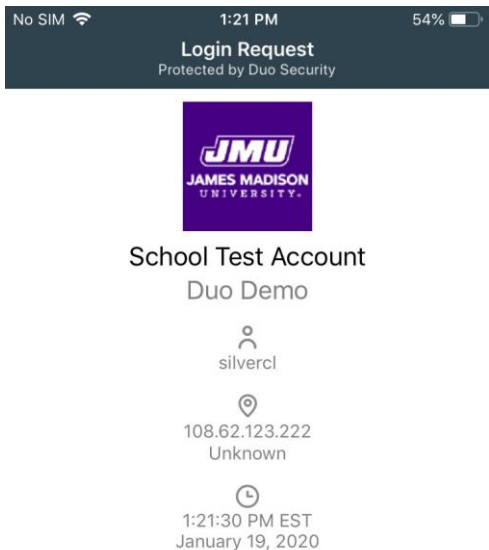


Figure 27 - Duo iOS App Push Notification Initiated by User Logging in with the Tor browser

The user is able to successfully authenticate to the application. It should be noted Duo does provide additional settings to prevent logins from anonymous networks such as TOR and VPN services.

VI. Conclusion

Existing two-factor mobile push notification based authentication systems have weak capabilities to help user's identify malicious logins in the case of real-time relay MITM phishing attacks. The UX and data points provided via the push notifications rely too heavily on an average user understanding networking concepts. Additionally, as of the time of writing it does not appear any system attempts to pull back useful data points about the client that is actually attempting authentication in order to make an evaluation that the current site the user is on is actually the one they intend. The PushValidator system described in this paper, has been shown to be able to detect real-time relay MITM phishing attacks against mobile push based two-factor authentication systems. The vast majority of modern two-factor systems in use are unable to detect such an attack but push notification based systems are well positioned to be able to. The PushValidator system shows how push notification two-factor systems can leverage their existing out of band communication channel and mobile apps to relay additional data points which can be used to help verify the client's connection to the server.

VII. Future Work

Future work could focus on determining ways to identify legitimate proxy and connection aggregators vs malicious real-time relay MITM phishing attacks since both exhibit extremely similar behavior. As described, this may involve whitelisting such legitimate proxies but there may be additional data points or connection properties which allow the target web application to discern a legitimate proxy versus a MITM phishing attempt. For example, there may be a way to fingerprint standard legitimate HTTP proxies based on connection metadata.

Another topic of interest would be further mitigating the effect of a compromised two-factor authentication provider. Discussed previously, if the target web application collects and distributes the user's mobile device public key then it can authenticate and validate device authentication responses independently of the two-factor service provider. This would mean even if the two-factor provider is compromised the target web application would be still be able to detect modified or spoofed device authentication responses. Additionally, to further expand such an approach, it may be preferable to completely remove the two-factor authentication provider from the equation and simply provide an SDK that contains all relevant functionality for the target web application to send push notifications, register devices and receive authentication responses.

Identifying additional authentication data points may be an interesting area to investigate as well. The additional data points could help reduce false positives and false

negatives in such systems. The data points may also be provide additional benefits with regards to threat hunting or may be useful in other security contexts. Research into novel ways of using the provided parameters to perform automated threat hunting and mitigation at scale could also yield benefits.

Glossary

To describe the attacks and proposed mitigation implementation some industry standard terminology is used along with some terms specific to the proposed implementation. The following definitions are provided to help eliminate any confusion.

Authentication data points – Data collected to help validate that the server the user is connecting to is in fact the target web application

MITM – Man-in-the-middle, meaning there is an entity between two communicating parties that is either able to view or manipulate the two parties' communications.

PushValidator – Described implementation to mitigate real-time relay attacks against mobile push notification two-factor systems.

Real-time MITM relay phishing attack – Scenario where an attacker emulates a target application by relaying requests and responses between the target application and the victim allowing the attacker to pass legitimate credentials to the target web application and upon successful authentication intercept persistent access cookies and tokens.

Target application – Application to which the phishing victim is attempting to authenticate

Two-factor [authentication] provider – Provides the target web application a means to send authentication requests which are then pushed to the user's mobile device and then is responsible for processing and storing the response

User – Client user attempting to authenticate to a particular application or service

User mobile device – Mobile device user receives push notifications from the two-factor provider on via the provider's app and is responsible for collecting the authentication data points

Victim – A user who has fallen for a phishing attack and had their credentials intercepted by an attacker.

Bibliography

- [1] M. E. Garcia, P. A. Grassi and J. L. Fenton, "Digital Identity Guidelines," US Department of Commerce, National Institute of Standards and Technology, 2017.
- [2] M. M'Raihi, S. Machani, M. Pei and J. Rydell, *TOTP: Time-Based One-Time Password Algorithm*, IETF, 2011.
- [3] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache and O. Ranen, *HOTP: An HMAC-Based One-Time Password Algorithm*, IETF, 2005.
- [4] S. D. M. O. C. S. B. L. C. N. C. Jessica Colnago, *"It's not actually that horrible": Exploring Adoption of Two-Factor Authentication at a University*, 2018.
- [5] Apple Inc., "Setting Up a Remote Notification Server," [Online]. Available: https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server#see-also. [Accessed 26 July 2019].
- [6] Apple Inc., "Sending Notification Requests to APNs," [Online]. Available: https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server/sending_notification_requests_to_apns. [Accessed 29 June 2019].
- [7] Verizon, "2019 Data Breach Investigations Report," 2019.
- [8] Reddit, "We had a security incident. Here's what you need to know.," 1 August 2018. [Online]. Available: https://www.reddit.com/r/announcements/comments/93qnm5/we_had_a_security_incident_heres_what_you_need_to/. [Accessed 22 June 2019].
- [9] B. Krebs, "Reddit Breach Highlights Limits of SMS-Based Authentication," 1 August 2018. [Online]. Available: <https://krebsonsecurity.com/2018/08/reddit-breach-highlights-limits-of-sms-based-authentication/>. [Accessed 22 June 2019].
- [10] World Wide Web Consortium, "Web Authentication: An API for accessing Public Key Credentials Level 1," 4 March 2019. [Online]. Available: <https://www.w3.org/TR/webauthn>. [Accessed 29 June 2019].
- [11] FIDO Alliance, "Universal 2nd Factor (U2F) Overview," 11 April 2017. [Online]. Available: <https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.pdf>. [Accessed 29 June 2019].
- [12] LetsEncrypt, "How It Works," [Online]. Available: <https://letsencrypt.org/how-it-works/>. [Accessed 8 July 2019].
- [13] Federal Bureau of Investigation, "Cyber Actors Exploit 'Secure' Websites In Phishing Campaigns," 10 06 2019. [Online]. Available: <https://www.ic3.gov/media/2019/190610.aspx>. [Accessed 08 July 2019].
- [14] M. A. P. T. Italo Dacosta, "Trust No One Else: Detecting MITM Attacks against SSL/TLS without Third-Party," in *Proceedings of the 17th European Symposium on Research in Computer Security*, Pisa, 2012.
- [15] I. Carrol, "Extended Validation Is Broken," [Online]. Available: <https://stripe.ian.sh/>.

- [Accessed 18 July 2019].
- [16] R. W. R. A. A. H. H. a. M. W. G. Adrienne Porter Felt, U. o. C. B. Christopher Thompson and E. M. a. S. C. G. Mustafa Embre Acer, "Rethinking Connection Security Indicators," in *Proceedings of the Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, Denver, 2016.
- [17] PhishLabs, "2018 PTI Report," 2018. [Online]. Available: https://info.phishlabs.com/hubfs/2018%20PTI%20Report/PhishLabs%20Trend%20Report_2018-digital.pdf. [Accessed 21 July 2019].
- [18] R. H. D. B. Rolf Oppliger, "SSL/TLS session-aware user authentication – Or how to effectively thwart the man-in-the-middle," *Computer Communications*, vol. 29, no. 12, pp. 2238-2246, 2006.
- [19] R. Abubaker, *Channel Based Relay Attack Detection Protocol*, 2019.
- [20] P. Knickerbocker, *COMBATING PHISHING THROUGH ZERO-KNOWLEDGE AUTHENTICATION*, Oregon, 2088.
- [21] M. A. a. P. T. Italo Dacosta, "Trust No One Else: Detecting MITM Attacks against SSL/TLS without Third-Parties," in *European Symposium on Research in Computer Security*, Berlin, 2012.
- [22] S. C. M. A. a. P. T. Italo Dacosta, *One-Time Cookies: Preventing Session Hijacking Attacks with Disposable Credentials*, 2011.
- [23] Google Inc., "Safe Browsing," [Online]. Available: <https://safebrowsing.google.com/>. [Accessed 7 October 2019].
- [24] D. Kennedy, "setoolkit," [Online]. Available: <https://github.com/trustedsec/social-engineer-toolkit>. [Accessed 20 11 2019].
- [25] R. G. Sadia Afroz, "PhishZoo: Detecting Phishing Websites By Looking at Them," in *2011 Fifth IEEE International Conference on Semantic Computing*, 2011.
- [26] J. Z. a. Y. Wang, "A Real-time Automatic Detection of Phishing URLs," in *2012 2nd International Conference on Computer Science and Network Technology*, 2012.
- [27] A. K. J. B. a. L. J. C. Zheng Dong, "Beyond the lock icon: real-time detection of phishing websites using public key certificates," in *2015 APWG Symposium on Electronic Crime Research (eCrime)*, Barcelona, 2015.
- [28] Microsoft Corporation, "Microsoft Authenticator on the App Store," Apple, 28 January 2020. [Online]. Available: <https://apps.apple.com/au/app/microsoft-authenticator/id983156458>. [Accessed 4 March 2020].
- [29] Duo Security, "Duo Web Two-Factor Authentication For Your Web Application," [Online]. Available: <https://duo.com/docs/duoweb>. [Accessed 10 January 2020].
- [30] P. Duszynski, "GitHub - drk1wi/Modlishka. Modlishka. Reverse Proxy," [Online]. Available: <https://github.com/drk1wi/Modlishka>. [Accessed 29 December 2019].
- [31] Microsoft, "Azure App Service Documentation - Azure App Service | Microsoft Docs," [Online]. Available: <https://docs.microsoft.com/en-us/azure/app-service/>. [Accessed 3 January 2020].

- [32] Lets Encrypt, "Lets Encrypt," [Online]. Available: <https://letsencrypt.org/>. [Accessed 11 January 2020].
- [33] C. Silver, "GitHub," 10 January 2020. [Online]. Available: <https://github.com/SilverC/DuoDemo>. [Accessed 10 March 2020].
- [34] C. Silver, "GitHub," 10 January 2020. [Online]. Available: <https://github.com/SilverC/PushValidatoriOS>. [Accessed 10 March 2020].
- [35] C. Silver, "GitHub," 10 January 2020. [Online]. Available: <https://github.com/SilverC/PushValidatorExtension>. [Accessed 10 March 2020].
- [36] C. Silver, "GitHub," 10 January 2020. [Online]. Available: <https://github.com/SilverC/PushValidatorDemo>. [Accessed 10 March 2020].
- [37] C. Silver, "GitHub," 10 January 2020. [Online]. Available: <https://github.com/SilverC/PushValidatorService>. [Accessed 10 March 2020].
- [38] Golden Frog, "VYPRVpn Official Website," [Online]. Available: <https://www.vyprvpn.com/>. [Accessed 10 January 2020].
- [39] Tor Project, "Tor Project," [Online]. Available: <https://www.torproject.org/>. [Accessed 11 January 2020].
- [40] Apple Inc., "Safari Extensions," [Online]. Available: <https://developer.apple.com/safari/extensions/>. [Accessed 16 October 2019].
- [41] Google Inc., "Create a deep link for a destination," [Online]. Available: <https://developer.android.com/guide/navigation/navigation-deep-link>. [Accessed 4 November 2019].