Zeropasswords

Patented

What's wrong with FIDO?

fido alliance | simpler stronger authentication

Nikos Leoutsarakos

Tiny bio
Nikos has a Physics background and a M.Sc. in Computer science from McGill University in Montreal, Canada, where he lives with his wife and two children. He has been publishing and developing systems in the areas of cryptography, wireless/mobility and digital signatures since 1994.

Nikos can be contacted at NIKOS@ZEROPASSWORDS.COM

Prelude

FIDO alliance (http://fidoalliance.org) has undertaken a noble mission to create a standard for online authentication. In its effort to include everyone the standard proposes two separate protocols named the Universal Authentication Framework (UAF) and the Universal Second Factor (U2F) protocols. While U2F aspires to become a second factor to usernames and passwords, UAF aspires to eliminate passwords. This document discusses and analyzes PKI-based challenge-response protocols in general and UAF in particular. A separate document discusses U2F and other protocols that augment username and password based authentication systems.

UAF is an instantiation of the PKI challenge-response protocol which originates from the '70s when PKI was first introduced. UAF modernizes the protocol and takes advantage of today's advanced mobile computing devices.

FIDO Registration

1. User is prompted to choose an available FIDO authenticator that matches the online service's acceptance policy.
2. User unlocks the FIDO authenticator using a fingerprint reader, a button on a second–factor device, securely–entered PIN or other method.
3. User's device creates a new public/private key pair unique for the local device, online service and user's account.
4. Public key is sent to the online service and associated with the user's account. The private key and any information about the local authentication method (such as biometric measurements or templates) never leave the local device.

FIDO Login

1. Online service challenges the user to login with a previously registered device that matches the service's acceptance policy.
2. User unlocks the FIDO authenticator using the same method as at Registration time.
3. Device uses the user's account identifier provided by the service to select the correct key and sign the service's challenge.
4. Client device sends the signed challenge back to the service, which verifies it with the stored public key and logs in the user.

UAF requires two (2) cooperating software applications: the first app runs at a Website and the second one runs in the mobile devices of its users. The Website app (FIDO server) has access to the public key of a user/device registered at the site. The mobile app (authenticator) has access to the corresponding private key. The Website app issues challenges to the mobile app and the mobile app sends back responses to these challenges. A challenge is a bit-string and a response is the same bit-string signed by the mobile app with the private key for the Website that issued the challenge. The Website app decrypts received response using corresponding public key and compares resulting bit-string to issued challenge bit-string. If bit-strings match, user verification succeeds, otherwise user verification fails.

UAF is secure because it keeps private keys private in mobile devices, public keys public at Websites, simple to implement and easy to use. What's wrong with that? Well, I found 15 things wrong with that!

1. **Weakest link**. All challenge-response protocols, including UAF and all the PKI-based ones, provide the same level of security dictated by the last step of the protocol which is a comparison of the response against the challenge. Regardless of the type of challenge used and the process employed to issue the challenge and receive the response, the weakest link of the protocol will always be the comparison operation that determines whether the received response matches the challenge. Hackers take advantage of this vulnerability and bypass challenge-response protocols altogether compromising only their final Boolean outcome (Yes or No) to their favor. Even the most sophisticated challenge-response protocol can be rendered useless by simply compromising the outcome of its last-comparison to always result in "Yes". Such a compromised challenge-response protocol would grant access to anyone!
2. **Sentinel process**. By design, UAF and all challenge-response protocols are agnostic to the service, system or asset that they protect, and initially they were heralded as the protocol that can "protect anything". This is achieved by designing the challenge-response protocols to stand separate and autonomous from the system they protect, with only a security token or trigger passed on between them. While this loose coupled architecture provides ultimate versatility to the protocol, it is at the same time its Achilles heel. The danger of misuse of generated security tokens, the danger of fabrication of fraudulent security tokens and the danger of unwarranted or compromised triggers are real and the subject of many exploits. Hackers know that if they can control the token or the trigger of a sentinel challenge-response protocol, they can control the system it protects.
3. **Chosen plaintext attacks**. A compromised Website, or a Website coerced by NSA for example, can discover the private key used by a mobile app simply by collecting enough challenge-response pairs over time and feeding them to a chosen plaintext

attack tool[1]. The same holds true for a man-in-the-middle who can also collect the same challenge-response pairs from an unsecure channel between the mobile app and the Website. UAF could have chosen a different challenge-response protocol that is not open to chosen plaintext attacks, but they didn't.

4. **Malicious challenge**. UAF stipulates that the issuer of a challenge will always be the Website server app, but does not specify a structure, form or restrictions on the content of the challenge. It also stipulates that the mobile app must sign the challenge with its private key and send it back to the Website. This simplistic protocol may cause mobile apps to sign "anything", including fraudulent document digests, fraudulent transactions, fraudulent approvals, etc. A user and his mobile app have no defences against a malicious or compromised Website whose challenges are randomly generated bit-strings or strings with unknown structure.
5. **Authenticator and authenticated**. Challenge-response protocols have been designed to allow only one of the parties to prove himself to the other party with the responder to a challenge always authenticating himself to the issuer of the challenge. If mutual authentication is desired, the challenge-response protocol must be executed twice, once for each party. Interestingly, UAF specifies only one execution of the challenge-response protocol in order to authenticate mobile apps to Website apps, deeming the reciprocal execution of the protocol unnecessary. The UAF assumption (not specification) that mobile apps will use TLS/SSL to authenticate Websites is poor because TLS authenticates Websites and not Website apps to mobile apps. Also, it relies on third parties (certificate authorities), and as FIDO warns, it is untrustworthy when a proxy is involved.
6. **A single key pair**. UAF is PKI-based and as such it limits itself to a single pair of public/private keys used to authenticate a single user device (private key), to a single Website, and to a single user account (public key). However, real life login situations often require more than a single pair of keys. For example, an online banking service or Website may allow several users to access the same account, or it may require the approval of a group of users in order to grant access to an account, or in other cases, it may impose an order or hierarchy on how multiple approvals are to be obtained. UAF is constrained by PKI which is strictly a single key pair technology and it is not clear in

[1] (http://www.cryptool-online.org/index.php?option=com_content&view=article&id=55&Itemid=53&lang=en).

UAF specs how Websites and online services will obtain multiple user authentications for a single account?

7. **Users and Websites are not liable**. UAF and all challenge-response protocols are unable to produce trustworthy transaction logs because responses, which are signed bit-strings, attest only to the private key that signed them and they do not contain any information about the Website, the user device, the holder of the device, the time and place of signing, and most importantly, they do not contain any information about the purpose or intent of signing. Consequently, maintaining logs of signed challenges is futile. Users have no record, receipt, or other type of proof in their mobiles, which would legally bind them, and their devices, to specific login transactions. At the same time, Websites also do not have a record or proof that legally binds them to specific transactions. The consequences of lack of login transaction logs are serious. Users are vulnerable and unable to prosecute or defend themselves in court against unlawful transactions that caused damage to their accounts, especially when they have been signing random challenges with their mobile apps. In that respect, UAF is no different than today's password based authentication systems which are unable to hold users and Websites accountable.
8. **Keys do not have expiry dates**. UAF specifies that public and private keys generated by the mobile app do not have an expiry date, i.e. they never change. Actually, UAF stipulates that keys should never change since the public key uniquely represents a user account at a Website and the corresponding private key uniquely represents the user/owner of the account. Clearly, the security of this architecture relies solely on private keys stored in user devices forever, and non-surprisingly, this architecture turns FIDO user devices into attractive targets for hackers and thieves.
9. **Private keys are not private.** Keeping private keys private is a paramount requirement of UAF and PKI-based challenge-response protocols in general. To satisfy this requirement UAF proposes the generation of private keys inside user devices and never allow private keys to leave their originating devices. Naturally, one would expect to find in UAF specs details on how private keys are linked to their originating devices, but there is no such mention. Instead, to avoid fraudulent copies of private keys into devices other than the originating ones, UAF suggests the use of a "monotonically increased counter" within each device so that a Website can detect cloned devices. This

would require a device to maintain a counter for each Website it is registered at, and it would also oblige a Website to maintain a counter for each one of its registered users/devices. Based on past experience, it is only time before such counters get out of sync and unintentionally cause denial of service.

10. **Public keys are in danger.** The public keys stored at Websites cannot be public. They need to be private, i.e. protected from public use, because UAF uses public keys to represent user accounts at Websites and must be kept in secure storage. Unprotected public keys may cause a user to be locked out of his account at a Website for ever, if a hacker manages to switch the user's public key with his own public key. This switch will allow a hacker to use the corresponding fraudulent private key and one of his (registered) devices to impersonate the user and access his account. Because this danger is real, UAF expects that Websites take measures to safeguard public key databases. Doesn't that remind you of the responsibility Websites have today to safeguard password databases?
11. **PKI cannot authenticate users.** One of the early criticisms of PKI-based challenge-response protocols was their inability to authenticate humans. These protocols can authenticate only keys in computing devices, and not their users or owners, because they rely on cryptographic challenges and responses which humans cannot compute. To resolve this issue, UAF stipulates that a user must first prove himself to his device[2] (e.g. fingerprint), then his device must prove itself to the Website (secure device identifier plus counter) and finally the mobile app must prove itself to the Website app (private key). The flaw in this three-step process is that the first step precedes the other two steps instead of being a prerequisite to the other two steps. According to UAF, user authentication is a local issue between a device and its holder and it does not provide a direct way (cryptographic or other) for Websites to know whether the holder of a device is the owner of the account being accessed.
12. **Service continuation.** UAF restricts login only to previously registered user devices which are entrusted with a private key. UAF also stipulates that private keys must never leave their originating device. Put these two stipulations together and UAF specs force a user to have only one device, or have a device per Website, or have some

[2] Device is short for "local device authenticator" in UAF terminology

Websites accessed by one device and some by another, etc. What if a user misplaces his device temporarily (forgets it at home); loses it, breaks it, or has it stolen. What if his device becomes inoperative because of faulty hardware, firmware or software? How is he to login then? Without redundancy the UAF protocol is severely handicapped.

13. **Authenticator attestation and status.** UAF stipulates that a user must always prove himself to his device using a local authenticator. UAF also warns that only authenticators meeting certification requirements defined by the FIDO Alliance and accurately describing their relevant characteristics will have their related attestation keys included in the default Trust Store. In other words, UAF forces Websites (i.e. FIDO servers) to receive regular updates of their trust store through an attestation service and be notified of compromised authenticators or compromised authenticator attestation keys. What this translates to is that authenticator manufacturers will have to inform FIDO alliance about compromised authenticators, and in turn, a FIDO Alliance attestation service will have to reach Websites and update their trust stores accordingly. FIDO Alliance goes to all this trouble to correct the UAF mistake of ignoring Websites completely when it comes to user authentication, making it strictly a local affair between him and his device.
14. **UAF is a 4-party protocol.** PKI does not authenticate users, it authenticates keys. To mitigate this problem, UAF is a minimum 3, and normally 4, party protocol. The parties are 1) the user 2) the Website 3) FIDO Alliance and 4) a certificate authority. The user needs a certificate authority to attest the validity of the key presented to him by the Website and the Website needs the FIDO Alliance to attest the validity of the user mobile app (local authenticator). The problem is that as soon as third parties are involved dependence and trust become issues. In order for FIDO to become a standard, the FIDO Alliance and all certificate authorities MUST be trusted by all users and by all Websites around the world! Do you see Chinese or Russian users and French or Turkish Websites, trusting American third parties (e.g. Verisign, FIDO Alliance)?
15. **User Consent.** Important transactions such as confirming a financial transaction, signing a legal agreement, or releasing patient records, cannot be silent or implicit and they require by law explicit user consent. Proof of explicit, non-coerced user intent to consent is paramount in the law (eSign Act). Unfortunately, UAF does not adhere to eSign act because its PKI challenge-response protocol cannot capture user intent with

just signed random bit-strings. Signed bit-strings do not contain information that can identify a user and a Website, the time & place where transaction took place, and most importantly they do not contain any information that can tie users and Websites to login transactions irrefutably.

Conclusion

"The U2F protocol lacks support for Secure Display, Transaction Confirmation, has only server-supplied Protocol Nonces, and Authenticator Class Attestation is implicit as there is only a single class of device". FIDO admits these issues in the security document of UAF specs. We added 15 more issues here. It is up to you now.