Fingerprint

Apple Engineering

Demo: Brute-forcing a macOS user's real name from a browser using mDNS

Konstantin Darutkin
Researcher and Developer

July 13, 2023

This article is the second in a series that explores potential privacy vulnerabilities in Apple devices. In the first article, we discussed detecting a system Apple ID region. This article presents a technique for revealing a user's first name without permissions using the mDNS protocol.

DISCLAIMER: Fingerprint as a company does not use this technique in our products, and we do not provide cross-site tracking services. We focus on detecting and preventing fraud and supporting modern privacy trends for removing third-party tracking entirely. There should be open discussions about such techniques to help internet browser providers fix them quickly.

Introduction

In this article, we explain how the real name of a macOS user can be leaked through a browser without permissions. The proof of concept demo is optimized for performance rather than accuracy and results may be affected by device or network configuration.

The name brute-forcing technique uses a pre-made list of the 50 most popular gender-specific names from a specific country origin. Our experiments showed that this is enough to detect a macOS user's name correctly in 65% of the cases on average.

Multicast DNS protocol and Apple Bonjour

The exploit implementation relies on the multicast DNS (mDNS) protocol. In simple terms, the mDNS protocol is designed to register, discover, or broadcast device names over a local network.

For instance, when a specific device, such as a printer, wants to be discovered on a local network, it sends a registration UDP packet to the reserved internal IP address `224.0.0.251`, which contains a hostname like `HP_LaserJet_Printer.local`. The `.local` domain TLD indicates that the hostname should be resolved using the mDNS protocol.

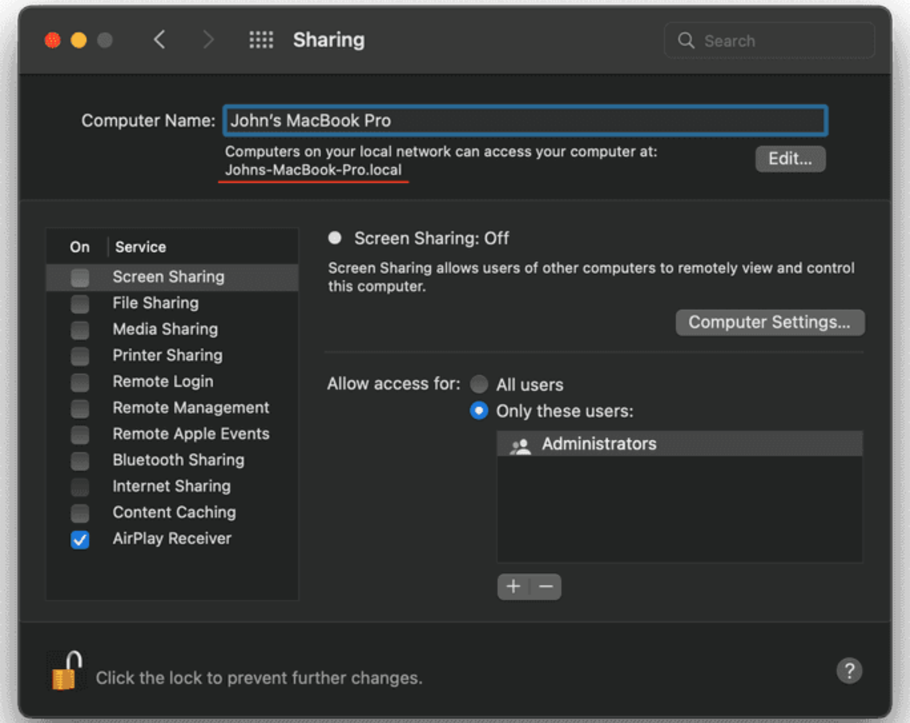
Such packets are automatically broadcast by a router to other devices in a local network, so they can cache the hostname. Alternatively, devices can send query packets to the same reserved IP address and try to discover a specifically named device, which may not exist in the network.

Some examples of mDNS hostnames are:

1. `johns-mac-mini.local`
2. `david-ZenBook-UX431DA-UM431DA.local`
3. `james-iphone.local`
4. `canon-mf644c.local`
5. `bedroom-appletv.local`
6. `dlinkrouter.local`

The multicast DNS protocol is widely used on Apple devices as part of the Apple Bonjour feature.

By default, Apple devices expose the first name of a user in their local hostnames, which we are going to use for the name brute-forcing technique. You can view or change your macOS local hostname in the **Sharing** section of **System Settings.**



Computer Name: John's MacBook Pro

Computers on your local network can access your computer at:
Johns-MacBook-Pro.local

- [ ] Screen Sharing
- [ ] File Sharing
- [ ] Media Sharing
- [ ] Printer Sharing
- [ ] Remote Login
- [ ] Remote Management
- [ ] Remote Apple Events
- [ ] Bluetooth Sharing
- [ ] Internet Sharing
- [ ] Content Caching
- [✓] AirPlay Receiver

Screen Sharing: Off

Screen Sharing allows users of other computers to remotely view and control this computer.

Allow access for: All users [✓ Only these users:]

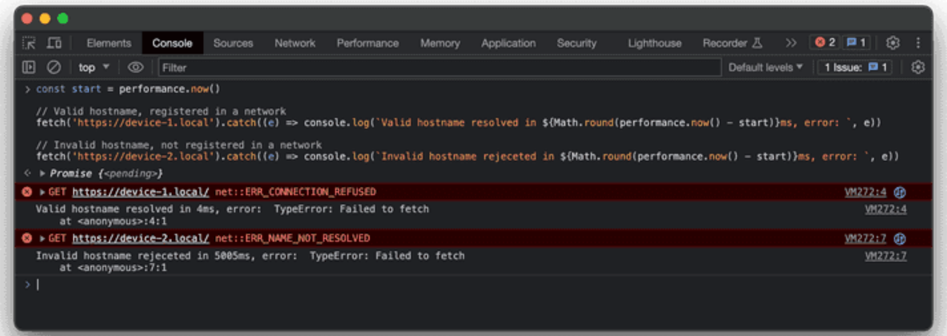
Administrators

Click the lock to prevent further changes.

Resolving mDNS hostnames from a browser

Unfortunately, the multicast DNS protocol is based on UDP packets. Browser JavaScript environments do not support arbitrary UDP sockets, so it is not possible to use the mDNS protocol directly in a browser.

However, we can resolve hostnames from browsers by using a timing workaround. Let's make two regular `fetch` GET requests to existing `device-1.local` and non-existing `device-2.local` mDNS addresses:



```
> const start = performance.now()

  // Valid hostname, registered in a network
  fetch('https://device-1.local').catch((e) => console.log(`Valid hostname resolved in ${Math.round(performance.now() - start)}ms, error: `, e))

  // Invalid hostname, not registered in a network
  fetch('https://device-2.local').catch((e) => console.log(`Invalid hostname rejeceted in ${Math.round(performance.now() - start)}ms, error: `, e))
< ▸ Promise {<pending>}
▸ GET https://device-1.local/ net::ERR_CONNECTION_REFUSED                    VM272:4
  Valid hostname resolved in 4ms, error:  TypeError: Failed to fetch          VM272:4
      at <anonymous>:4:1
▸ GET https://device-2.local/ net::ERR_NAME_NOT_RESOLVED                     VM272:7
  Invalid hostname rejeceted in 5005ms, error:  TypeError: Failed to fetch    VM272:7
      at <anonymous>:7:1
> |
```

The browser will try to resolve the hostname provided in a URL address. If the address is resolved, it will send a TCP packet to the 80 port, which in our case will most likely be closed. On the screenshot above you can see two different error messages:

- `ERR_CONNECTION_REFUSED` for the existing `device-1.local`
- `ERR_NAME_NOT_RESOLVED` for non-existing `device-2.local`

Both errors will be mapped into the same `Failed to fetch` JavaScript error, so we can't rely on the error type, but we can perform a timing attack. Local networks are fast, so the valid mDNS hostname registered in the network will be resolved in a reasonable time frame, which is significantly faster than the default connection timeout. In the example above, the difference is four milliseconds for a valid address versus five seconds for an invalid one.

This approach is consistent enough for the proof of concept solution and works similarly in all major browsers. In practice, you can use any network JavaScript API, such as `iframe`, `Image` or `WebRTC`, to perform timing attacks for DNS resolving.

MacOS user name brute-forcing

As illustrated earlier, the default macOS local hostname contains the user's first name and device name. Moreover, the hostname depends on a system language locale:

- English: `<name>s-macbook-pro.local`
- French: `macbook-air-de—<name>.local`
- Russian: `mac-mini—<name>.local`

For example, we can take the top 1,000 names, the top 10 locales, and five common macOS device names. In this scenario, it would be necessary to test 50,000 distinct hostnames, which might take over an hour. A more efficient strategy would be to limit the search scope to a single locale, a single device, and the 50 most common names within that specific locale. While this affects accuracy, it makes the attack more feasible in practical terms and significantly faster in general.

The locale selection can be based on a browser time zone, language, or IP address location. Safari browser, for example, reveals the system locale with the `navigator.language` property, which is typically consistent with the targeted hostname locale. Also, there are other workarounds to discover the user's country of origin, such as the Apple ID region detection method discussed previously.

The device options can be narrowed down by using the screen resolution. For instance, the `1728x1117` resolution is most likely a 16-inch Macbook Pro. An extended screen can be detected by using the `screen.isExtended` property, which will fallback the device options to three to five of the most commonly used Apple macOS devices.

The proof of concept demo has a name country origin and name gender selectors, which results in 50-100 potential hostnames that can be enumerated within seconds. Among the 100 FingerprintJS employees who participated in an internal test, the demo successfully predicted the name in 65% of instances. The source code is available on GitHub.

It's important to mention that the proof of concept demo is simply a demonstration of the attack and may not be successful in some cases. Some of the factors that could impact the results are:

- The presence of rare or unique names or non-standard macOS hostnames
- Particular network configurations or VPN networks
- Enabled firewall in macOS network settings

If the demo does not work for you, consider using the advanced configuration settings to verify the list of the probed names and the device name pattern applied. If you use a VPN, consider testing it with and without VPN, because the result may change.

Conclusion

Considering the inherent weaknesses and numerous limitations, this attack isn't practical. It can be effortlessly detected in the network tab of browser developer tools unless there is deliberate intent from a website owner to de-anonymize its visitors.

This series of articles merely explores the boundaries of internet privacy and relies on unconventional privacy breaching techniques. As another example, by combining this method with detection of installed applications, there’s a potential to develop a harmful website capable of displaying your real name and job title, based on the list of professional applications used, all without requiring any permissions.

Even though this article mentions Apple devices running macOS, the mDNS discovery technique can be utilized in variety of ways. For instance, it could be used to perform a local network scan to detect devices such as printers, smart TVs, smart speakers, and other home IoT devices.

This method is also applicable to iPhones and iPads, given that sync over Wi-Fi or Safari remote debug features are activated.

All article tags

Apple Engineering

Related Articles

April 11, 2023

How Smart App Banners can be used to reveal Apple ID region

Discover a new privacy vulnerability affecting Apple iOS users and learn how your Apple ID country or region can be detected without your permission on Safari for iOS and iPadOS. Stay informed and protected with our informative series exploring potential privacy issues in Apple devices.

Apple Leak