WebXR Fingerprinting

Daniel Pagan
Information Networking Institute
Carnegie Mellon University
Pittsburgh, U.S.
dap2@andrew.cmu.edu

Tristian Pittman
Heinz College
Carnegie Mellon University
Pittsburgh, U.S.
tdpittma@andrew.cmu.edu

Venkata Kanthuru
Information Networking Institute
Carnegie Mellon University
Pittsburgh, U.S.
vkanthur@andrew.cmu.edu

***Abstract*—With virtual experiences in VR and AR on the rise, the security and privacy of this technology should be a major concern. This paper explores the online tracking implications of the experimental WebXR browser API and relates it to the established field of fingerprinting. We explore the API, documenting lessons learned and building proof-of-concept adversaries and defenses for fingerprinting in this context. We show how emulated VR and AR devices are uniquely identified and how this data can be collected for future analysis and correlation against other data sets. We find that WebXR provides a low-effort way to fingerprint VR and AR users and caution both users and browser vendors to these types of techniques.**

***Index Terms*—Fingerprinting, WebXR, Browser Security, Privacy, Tracking**

I. Introduction

Browser fingerprinting is a technique that websites use to uniquely identify a particular browser user and track them across their browsing session. Malicious actors practice subverting user defenses and still collecting uniquely identifiable information about users.A trivial form of fingerprinting is through the use of cookies which are set during first visit and reused across different sites. Cookies are easily disabled or blocked by users. There are even automated extensions that block known tracking cookies or tracking domains [1]. It turns out that trackers can use different internal peculiarities of a browser to uniquely identify users even if they disable cookies.

Tracking and fingerprinting are an integral part of the digital advertising industry because it enhances advertising and tracking precision. The digital advertising industry in the United States is expected to hit $278 billion by 2024 [2]. Tracking and fingerprinting also help third-party entities like advertisers collect information that users do not intend to share. Fingerprinting can be seen as an invasion of privacy and organizations like the EFF fight against tracking and fingerprinting [3].

Another interesting area of computing that has been on the rise is virtual experiences. Virtual Reality (VR) and augmented reality (AR) technologies have been growing exponentially over the last couple of years. AR alone is expected to become a $77 billion industry by 2025 [4]. Companies like Meta have even re-branded to reflect the industry interest in virtual experiences. Interest lies in changing how we work through virtual meeting spaces, how we play through video games, and how we socialize through events like virtual movie nights. There has even been documentation of people living their lives through VR, only taking a headset off for the essentials [5].

With the interest in virtual experiences, a specification has been proposed to allow for a common browser interface to VR or AR devices [6]. This specification details the WebXR API, which has already been adopted in both Firefox and Chrome although being labeled as experimental. The API allows for developers to interact at a high level with both AR and VR devices while they are attached to a normal computer. Users navigate the web normally until they hit content with a virtual experience. From here, they can put on their device and interact with the content. The WebXR API is purely accessible from JavaScript and allows developers to set up these experiences with traditional web technologies.

This paper lies at the intersection of these two technologies. We bring fingerprinting techniques to the WebXR API. We believe that there is uniqueness to fingerprinting VR or AR devices and that this area is important for the privacy of the future of the web. Because both the fingerprinting and VR/AR industries are on the rise, we think that advertisers and tracking companies will soon move into this space.

The rest of our paper illustrates our explorations into fingerprinting using the WebXR API. We explain lessons learned, interesting features, and security/privacy considerations. In Section III we describe the adversarial model that fingerprinting poses to the WebXR API. In another Section IV we show our work implementing fingerprinting for the API and show a proof-of-concept implementation of an adversary in our threat model. In Section V we explore possible defenses against WebXR fingerprinting and share a chrome extension that can help to prevent access to the WebXR API during runtime. In Section VI we explore related work in both the fingerprinting and VR/AR spaces. Finally, we conclude in Section VII.

II. Methodology

A. Timeline

This research study that our team completed had a deadline for completion, as such we had to accomplish certain items in a organized manner. We decided to complete the different parts of this study in four phases which are broken up into two to three weeks.

1) Review Phase: This is where we reviewed literature on the subject, looked for existing solutions for tracking

or defending, and decided upon priority-1 vs priority-2 goals.
2) Unique Phase: The next phase was to figure out the uniqueness of the project, we wanted to add literature to a subject in fingerprinting which had not been considered as much.
3) Implementation phase: This establishes how fingerprinting is done with WebXR and we record our findings. Our goals for implementation were to:
   - ☛ Understand source code
   - ☛ Document interesting findings
   - ☛ Quantify Unique Identifiers
   - ☛ Find Defenses or Mitigations
4) Conclusion Phase: Bring together all the interesting findings and inferences that were drawn from the data to finalize a report.

B. Research Questions

1) How is fingerprinting achieved through WebXR?
2) How can one prevent fingerprinting of AR/VR devices?

C. Data

All data was recorded in a spreadsheet, and the more interesting and important findings are provided in a table discussed within the Evaluation subsection of IV. Fingerprinting Web Server.

III. Threat Model

In order to begin understanding fingerprinting in relation to WebXR, we need to first build up a model of the technologies and adversaries. We define the basics of fingerprinting and fill in background on the area of research. From here, we describe the threat model we used in our research of the WebXR API. We also define two different types of adversaries that are relevant to the real-world implications of fingerprinting and tracking in VR or AR.

A. Fingerprinting Basics

Because fingerprinting is mainly an invasion of privacy, assessing the threats and building a model based on these threats has to be examined a bit differently than in regards to traditional threats. It is stated that "The goal of tracking one's browsing habits is usually to amass as much information about her from as many sources as possible. Such an extensive collection of data about somebody is called profiling"[7] This is because most fingerprinting is conducted by the browser itself, and that data must be requested by web-pages and third-party applications in order to be used in malicious or commercial ways. The real-world environment consists of websites with myriad third-party content scripts included, which could be collaborating to get a better picture of the user's browsing habits.

There are multiple types of browser fingerprinting that can conducted: Browser-Specific fingerprinting, Canvas Fingerprinting, JavaScript Engine Fingerprinting, and Cross-Browser fingerprinting. [8] Browser specific fingerprinting is dependent on the browser environment, canvas fingerprinting relies on the canvas tag and image rendering in HTML, JavaScript engine fingerprinting is where JavaScript conformance is tested. Finally cross-browser fingerprinting is described as "the technique where the fingerprint is independent of the browsing environment. A subcategory of this technique is Accelerometer Fingerprinting, it is the use of a mobile device sensor and its calibration errors for the generation of a fingerprint." [8]

All of these fingerprinting techniques build up a picture of a user. Alone they may not uniquely identify the user, but together a tracker with sufficient data collection capabilities could match the union of all of a user's fingerprints and uniquely identify their network traffic. This spans across websites through third-party scripts. We will see that this is taken into consideration in our adversarial models that follow.

B. WebXR Threat Modeling

In the case of WebXR, the basic privacy threats do not differ much. In general the types of fingerprinting that are found with these devices are in Browser-Specific fingerprinting, and Cross-Browser fingerprinting. This is because the website maintains an active fingerprint of the VR/AR device, along with various details which give the indication of the type of device, such as whether or not the device has controllers. The attackers or "*fingerprinters*" are likely going to fall into one of three different categories: fingerprinting companies and third-parties, advertisers, or malicious trackers which are the least likely option.

WebXR devices are particularly vulnerable to fingerprinting because they do not require explicit confirmation to determine whether or not fingerprinting can or cannot be done. This means that "*fingerprinters*" that are able to successfully request uniquely identifiable data can do so without the user knowing anything. WebXR fingerprinting is especially interesting because of all the different unique identifiers that can be discerned by the browser per device. For example by gathering some of these identifiers "*fingerprinters*" can obtain positional data of the user or the space they are using, determine whether or not the device is in use. The types of fingerprinting data that is being conducted and benefits of the said fingerprinting done would vary by the type of "*fingerprinters*" for example if fingerprinting is done for commercial uses then knowing the type of device, how often it is being used, and metrics like what a users height might be could determine the type of ads that one might be targeted for.

In Fig. 1 the WebXR threat model is depicted. A user has their XR device plugged into the computer, which is running a traditional browser such as Chrome or Firefox. This browser implements the WebXR API. During normal web browsing, the user visits an attacker's website. This visit could be through natural browsing or through something more targeted like a phishing scam. After the user visits the attacker's site, the attacker ships malicious WebXR code to the browser which fingerprints the user and sends the data back to the attacker. The attacker is now free to perform data analysis, track the

user across differing sites, or to correlate their fingerprint with other data sources.

Adversaries in this threat model could be running their operations through first-party websites that victims visit like described before. They also may be modeled as an adversary like an advertiser, which would mean that they ship a third-party script that would be rendered in an advertising platform across many websites. This technique would allow the adversary insight into many different areas of the internet and allow them to more accurately track users. This is the most common threat model seen through advertising trackers on the internet.

It is worth to note that our model does not cover non-traditional browsers. Browsers that do not implement the WebXR API are obviously out of scope for this paper. Another interesting portion of related browsers that we do not consider is browsers that are integrated into the native operating system of the VR/AR devices themselves. That would require a separate threat model and be similar to testing embedded browsers. While out of scope for this paper, other work should investigate these embedded VR/AR browsers.

IV. Fingerprinting Web Server

Our fingerprinting web server shows a concrete implementation of the adversary in our threat model. We utilize Mozilla's WebXR emulation chrome extension [9] to emulate different VR/AR devices for our testing. For a subset of the VR/AR devices that exist in this emulator, we show that simple techniques can achieve success in fingerprinting without user interaction. There are many security and privacy concerns that should be considered with this technology. We also evaluate our solution in fingerprinting our initial device list. Finally, we show that there are still limitations with our approach.

A. WebXR API Internals

To start our exploration of the fingerprinting web server, we need to understand how the WebXR API works. The combination of knowledge about the implementation of the API and the fingerprinting mindset, leads to new insights about the design of the API. In order to show that a malicious fingerprinting adversary with script execution on the page can use the WebXR API to achieve their goals, we need to explore the inner workings of the API. The WebXR API is accessed through the navigator child of the window object. The navigator contains information about a visitor's browser and is a common parent of many fingerprinting APIs. Fig. 2 shows what the xr child of the navigator gives us. It provides access into all things WebXR.

Figure 2. XRSystem object returned by navigator.xr.

Using this object, we can request a session to the XR device (which can be either a VR or AR device). Sessions represent an abstraction over a connection to the device and we must specify the type of session we wish to receive. The following are the types of sessions that are available to a requesting script:

- inline - These devices are forced to be implemented by default for any device. They also cover a session type that is typical for streaming 3D video. An example of this would be the inline VR mode for YouTube videos when you are viewing on a normal computer without a VR/AR device plugged in to your computer.
- immersive-vr - These kinds of device sessions represent a full VR experience where the user is fully immersed into the VR world and the real world does not play a part in their experience. This would be the session used for most typical VR headsets such as the HTC Vive.
- immersive-ar - These kinds of device sessions represent any augmented reality experiences. These mesh the real and virtual worlds. Typical devices include Samsung's Gear AR and Microsoft's HoloLens.

Sessions give a window into the user's current device and environment. They give a developer access to the viewing pose, the user's surroundings, the status of their equipment (controllers, gamepads, etc.), and more. Most of the interesting functionality of the WebXR API lies behind a valid session.

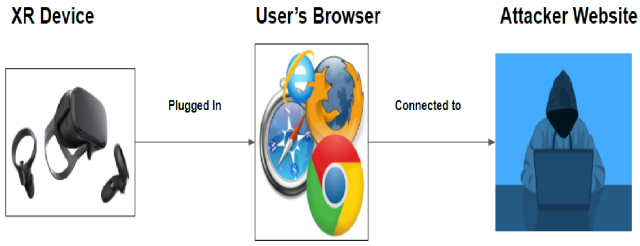

Figure 1. Fingerprinting Threat Model WebXR

These functions and attributes are the focal point of fingerprinting through WebXR. They can be seen in Fig. 3.

```
XRSession {visibilityState: 'hidden', renderState: X
ources: XRInputSourceArray, domOverlayState: null, d
  depthDataFormat: (...)
  depthUsage: (...)
  domOverlayState: null
  environmentBlendMode: "opaque"
▶ inputSources: XRInputSourceArray {length: 0}
  interactionMode: "screen-space"
  onend: null
  oninputsourceschange: null
  onselect: null
  onselectend: null
  onselectstart: null
  onsqueeze: null
  onsqueezeend: null
  onsqueezestart: null
  onvisibilitychange: null
  preferredReflectionFormat: "rgba16f"
▶ renderState: XRRenderState {depthNear: 0.1, depthF
  visibilityState: "hidden"
▶ [[Prototype]]: XRSession
```

Figure 3. XRSession object returned after requesting one from XRSystem.

Armed with a valid session, we can now find interesting portions of the session to use for fingerprinting. One such part of a session is the set of input devices. The specification [6] allows for input sources that range from handheld devices to gaze-based input. All of these are enumerable from an active session. In our investigation we found that not only can we poll the presence of buttons (both press and squeeze buttons) on handheld input devices, but we can also pull the manufacturer's identifier of the device. We will see later in our evaluation that the manufacturer identifier was invaluable in fingerprinting devices.

There are several extensions to the base WebXR API that allow for more advanced functionality. These can be used to fingerprint beyond just the type of device. One interesting API is the Hand API. It allows for a user to share accurate data from each joint in their hands. From a fingerprinting perspective, this could allow for more of a hand-print to be developed where a fingerprinting script could differentiate not only between different devices but also differentiate between whatever user of the machine is currently browsing the web. Fingerprinting the person rather than the browser aligns much better with the goals of our adversaries that we consider in this research and thus makes it a highly sought after piece of functionality. Fig. 4 shows the joints that are accessible to a developer.

Another optional API that is present that we considered for this research was the Depth API. This API allows a developer to query the distance between the headset and any physical objects in the user's room. A fingerprinting script could use

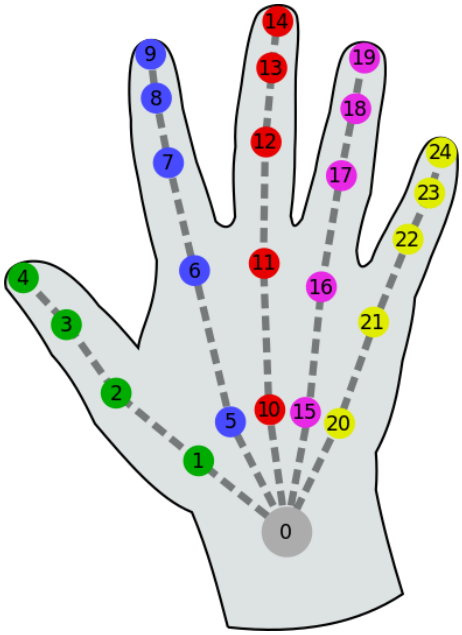

Figure 4. Joint model accessible through the Hand API.

this to gather information about the room that the user is currently browsing in. An example would be a user that is browsing on their laptop in the living room with a VR device and then later moves to the bedroom. The adversary is then able to detect that move and use the information for nefarious purposes such as targeted advertising.

B. Implementation

In order to get a practical implementation of WebXR fingerprinting we develop a malicious website that collects WebXR information and stores it for future analysis. Fig. 5 shows our architecture model. This is composed of a Node server [1] that hosts an endpoint that collects fingerprints. It also serves static content which contains our front-end fingerprinting code. Users visit our site and see a VR game. Without prompting them at all, we automatically collect all of the WebXR information the API exposes.

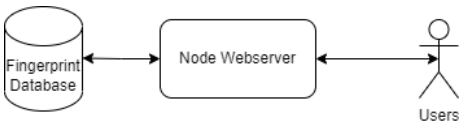

Figure 5. Architecture diagram for fingerprinting web server.

When a user connects to the malicious website, our fingerprinting script is loaded. Interestingly, when we called our script by the name fingerprint.js our ad blockers (in this case uBlock Origin [10]) were preventing it from loading. The fingerprinting script then starts a WebXR session and records all the information described in the previous section. This is wrapped into an object which is serialized and sent to the web server asynchronously at its fingerprinting REST API.

On the server side, we de-serialize the received fingerprints and write the fields into a CSV file for analysis. The choice of dumping to a CSV file was purely due to the small scale of our testing and for proof of concept. A real adversary or tracker would collect these in a more scalable solution or perform further processing on the back-end like correlating data between different data sets.

C. Evaluation

As mentioned previously, we used Mozilla's WebXR emulation extension in order to simulate different VR and AR devices. In order to evaluate our implementation we connected with each different device provided in the extension and recorded the fingerprint. Table I shows the results and illustrates that we were able to uniquely identify each of the devices in the extension.

Our implementation properly shows the adversary shown in our threat model. It serves seemingly benign content with malicious tracking embedded in it. It requires no interaction from the user and happens invisibly to the normal user. Through our fingerprinting script that runs in the DOM we show that we can exfiltrate the fingerprinting data to any domain, which confirms that third party scripts could perform this fingerprinting and share the data with a collection server. WebXR adds a large platform for fingerprinting and our implementation only scratches the surface of what can be done.

D. Privacy Concerns

Fingerprinting enhances the tracking and advertising industries. With our research into WebXR tracking we potentially open up new vectors for fingerprinting and tracking that users may be unaware of. We caution not only users about the dangerous possibilities of WebXR but also browser vendors who may be thinking of implementing the spec in their browser. Given how simple it was to get unique fingerprints through our implementation, we caution browser vendors to consider some simple defenses.

Later we show a way to prevent WebXR from being accessible in the first place. Another simple fix would be to prevent device specific strings from being leaked through the API. Our results found unique fingerprints mainly because of the controller identifiers from the manufacturers being too specific to the controller and device.

E. Limitations

Our research is not without limitations. Our limitations stem mainly from using emulated devices. Emulated devices are obviously not the real thing and therefore we cannot confidently say that our findings apply directly to actual VR or AR devices. One limitation from this is also that we are unable to emulate devices not implemented by our emulator. We cannot make any distinction between a device we have seen and one we have yet to test with our infrastructure. This is a problem of hardware accessibility to our research team and not a limitation to a sufficiently powerful adversary.

Another interesting limitation is customized environments. Users of VR or AR devices may not have factory default hardware. Custom input devices like hand or body trackers may not return unique information like manufacturer identifiers. Like the previous limitation, we cannot say anything about custom environments like this without access to that hardware.

A final limitation for this portion of our research is the failure to access optional portions of the WebXR API. Our emulated WebXR extension did not support the hand API or the depth API and we were therefore unable to produce a proof of concept implementation of fingerprinting with those interfaces. Despite this, we still consider these to be relevant fingerprinting threats. Future research with appropriate hardware should consider testing these interfaces in a fingerprinting context.

V. Defense

Defending against fingerprinting is a complex trade-off. On one hand, we would like to preserve a user's privacy and security as they browse the internet. On the other, tracking companies have great sway over browser vendors. Developers also enjoy the convenience of abstractions over complex functionality, like WebXR provides over VR and AR devices. Similarly, users want to get to their content as quickly as possible, making security dialog boxes inconvenient. For the purposes of our paper, we consider defense from both the perspective of a security-conscious user and the perspective of a responsible browser vendor.

The WebXR specification outlines several interesting security parameters. It allows requests to a VR or AR session based on if certain checks are passed. One of them is the user intent where an action is carried out on behalf of a user's actual action [11]. The specification outlines this so that code from the internet can not spoof user actions. Where this gets interesting in the context of WebXR implementations is that the browser assumes that if a website's code runs at launch, it is acceptable user intent. This means that if any fingerprinting takes place at the launch of the website, the browser allows this code to run.

Another security check defined by the specification is that the launching session must be triggered by responsible code. While this may seem like an abstract notion, the specification elaborates on what this means for the WebXR context. The website trying to launch a VR or AR session must be active and focused by the browser.

Given that these security mechanisms are easily met by most tracking or advertising code that runs in browsers, there needs to be stronger defenses. We approached this problem from two angles. First, we implemented an extension that prevents execution of such calls by default. The following subsections describe this extension in detail. Second, we took our insights from diving deep into the WebXR specification and make recommendations for solutions that we were unable to implement.

Table I
FINGERPRINTED DEVICES

Device	Input Devices	XR Session Type	Button Support	Headset Position	Controller ID	Unique?
Google Cardboard	0	Inline, VR	None	No	None	Yes
HTC Vive	2 controllers	Inline, VR	Select, Squeeze	Yes	OpenVR Gamepad	Yes
Oculus Go	1 controller	Inline, VR	Select	Yes	Oculus Go Controller	Yes
Oculus Quest	2 controllers	Inline, VR	Select, Squeeze	Yes	Oculus Touch	Yes
Samsung Galaxy S8+	2 controllers	Inline, AR	None	Yes	Empty String	Yes
Samsung Gear	1 controller	Inline, VR	Select	Yes	Gear VR Controller	Yes

A. WebXR Pop-Up Extension

A known prevention technique for complex browser functionality that is rarely accessed is to have a pop-up dialog box display before that subsystem becomes active within the browser. This can be seen in modern browsers through users having to allow access to their microphone for a website that requests it. WebXR is a similar technology that we argue should also be protected by a dialog box. Given our research time and our resources, we realized that extending a browser would not be viable for this paper. We decided upon a browser extension that detects whenever a WebXR session is requested and allows the user to deny the request graphically.

In order to design our extension, we must lay out requirements for it. Our requirements for this extension were to block all VR or AR session requests and provide a layer of consent that the web page must ask for through a popup dialog box. They can then make a choice based on whether the website actually serves the website's VR or AR content. If there is no VR or AR content on the page, then it is most likely trying to use it for fingerprinting purposes and the user can deny access. Providing an extension that meets these requirements gives power back to users to deny fingerprinting during their browsing experience.

Ideally this extra functionality would be implemented in the browser itself. By choosing an extension we actually increase the surface area of attack within our model. Consider a fingerprinting adversary that knows of the existence of this browser extension. The adversary may now use the pause in the WebXR request timing to collect an extra data point that the user has the extension installed. This applies even if the user decides to not enable the WebXR request. Despite this downside to our threat model, our team still believes that this extension moves WebXR fingerprinting privacy in the right direction. Until a browser based dialog box is implemented, we recommend blocking WebXR requests like our extension does.

B. Implementation

The extension works by wrapping the requestSession API [12] call with our code. Anytime a website tries to request a new VR or AR session, they run our wrapped code which raises an alert box that asks the user if they want to give the website access to the API. Fig. 6 shows the popup that is presented to users when either they or a website requests access to a WebXR session.

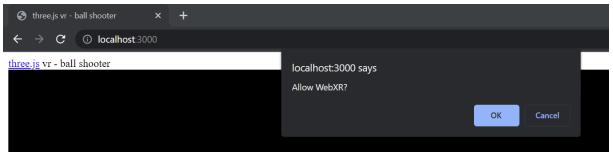

Figure 6. Extension Popup.

In order to show that our extension successfully denies fingerprinting data collection, we pitted our extension against our fingerprinting web server code described in the previous section. Fig. 7 shows the log from our web server fingerprinting code when the user denies access.

```
Got fingerprint [object Object]
{ has_xr: false }
```

Figure 7. WebXR API fingerprint denied by our extension.

The web server is only able to see that WebXR has not been enabled as the call to requestSession fails. On the other hand, if a user accepts the dialog prompt that allows the webpage to access the WebXR requestSession API, the web server is able to generate the full fingerprint. Fig. 8 shows the logs on the web server when the user allows access. This is treated as a normal operation of the WebXR API.

```
Got fingerprint [object Object]
{
  has_xr: true,
  supported_session_types: [ 'immersive-vr', 'inline' ],
  'immersive-vr': {
    num_controllers: 2,
    handset_id: 'OpenVR Gamepad',
    handset_has_position: true
  },
  inline: {
    num_controllers: 2,
    handset_id: 'OpenVR Gamepad',
    handset_has_position: true
  }
}
```

Figure 8. WebXR API fingerprint allowed by our extension.

A convenient technicality of implementing this extension was that the WebXR API's requestSession function call returns a JavaScript promise. By wrapping that code, we still return a promise and provide an invisible change to the functionality of the code from the perspective of a WebXR content developer.

One of the challenges associated with building this extension is that the content scripts of an extension run in an "isolated" world [13]. Content scripts are pieces of JavaScript code that an extension injects into contexts that have access to some version of the browser DOM. Isolated worlds prevent the modification of the web page's JavaScript environment. An isolated world in this case references a browser's isolation mechanism that prevents scripts from sources other than the website (such as an extension in our case) from accessing each other's JavaScript contexts. This means that our anti-fingerprinting extension's wrapping of the WebXR requestSession function is not visible to the web page code that we wish to target. Any changes we make are useless as the original API is still accessible to the web page, rendering our anti fingerprinting extension useless.

To overcome this, we have to force the extension's injected code to run in the same context as the web page. This is done by executing our anti-fingerprinting function in the page by setting its world to "MAIN" [14]. The main world in this case references the JavaScript context that the website code runs in. By making this change, any changes made by our extension to overwrite functions is visible to the web page. Now, any access to the requestSession API by the web page will result in our wrapped code being run.

As seen in Fig. 6, our code simply polls the user for a confirmation dialog. We decided to use the built in browser confirm dialog box to serve this function. In the next section we will see how this could be improved and what other choices we would have made given a deeper dive into how browser dialog boxes work.

C. Limitations

With so many dialog boxes showing messages with security information that are not particularly understandable by users, most users just click through them. Our extension faces the same limitation. If the user doesn't understand the implications properly and allows web pages access to the VR or AR APIs, then they might be fingerprinted. Further research should be done in this space to find the most effective pop-up dialog that maximizes user caution.

Additionally, if the page has valid VR or AR content along with fingerprinting utilities, the user might grant access thinking that the web page is trying to show them content when instead it might be fingerprinting them. Identifying whether a particular call to requestSession is for the web pages content or for fingerprinting is beyond the scope of our extension. Future research could explore this space by automatically identifying fingerprinting code from normal WebXR code using some form of classification model.

Finally, since the browser does not provide guarantees about when the extension is exactly run, race conditions might occur. A race condition occurs when two distinct computations are racing for a shared resource. This race is between our extension and the web page to see if we can wrap the call before their fingerprinting code can run. In the case of our extension, we set our code to be injected into the MAIN world as early as possible. This does not, however, cover the implicit trust that the WebXR specification currently applies to user launched websites. These sites are allowed to run their fingerprinting code upon startup rather than hidden behind an "enable VR" button. This is our race condition where our extension may or may not prevent fingerprinting. We believe that an extension is not the right choice of defense and that a browser vendor should implement this directly into the browser to prevent this race condition.

D. Recommendation

Determining the right defense against fingerprinting is difficult. Any solution will be unable to perfectly differentiate fingerprinting from legitimate WebXR functionality. Given this challenge, we think that user input is a good first step. We tried our hand at adding this functionality with our extension. Solutions to this problem can be viewed from two perspectives: the browser vendor and the user.

A recommendation to browser vendors would be to include a permission set for VR or AR API usage similar to how users are asked whether they want to allow a website access to resources such as their camera and microphone. The timing issues of our extension would no longer be applicable.

User's in the meanwhile can install plugins such as ours that prevent access to similar APIs. While it is not a fool proof mechanism, it still adds some form of protection. First, it prevents the case where WebXR code is run by default. Irrespective of whether the user is capable of always correctly discerning a website's use of the APIs, asking them to opt-in manually is more likely to prevent unwanted usage. Second, it raises awareness on the various ways organizations can fingerprint you and hopefully prompt users to ask for legislation on improving their privacy in a world where everyone wants to know what someone is doing.

VI. Related Work

As we reviewed various articles and research papers we compiled a list of references which we utilized to further support our findings and broaden our understanding of WebXR as it pertains to browser fingerprinting and the current capabilities associated to this API.

A. WebXR API

WebXR is still a relatively new technology and as such there is not a large volume of information which covers this subject. The creators at MDN describe WebXR in the fundamentals section as:

> "an API for web content and apps to use to interface with mixed reality hardware such as VR headsets and glasses with integrated augmented reality features. This includes both managing the process of rendering the views needed to simulate the 3D experience and the ability to sense the movement of the headset (or other motion-sensing gear) and provide the needed data to update the imagery shown to the user. WebXR additionally provides support for accepting inputs from control

devices such as handheld VR controllers or specialized mixed reality gamepads."[4]

Two papers in particular did thorough explorations about WebXR technology and gave evaluations based on the capabilities that are present.[15], [16] These two papers gave our team insight as to what types of development kits were in use, the vital features provided by the API, the precision of the API, use cases of the capabilities, and the areas that the WebXR excels. These papers also deliver limitations, design issues, and future challenges that they perceived after thorough reviews of the contemporary WebXR capabilities. These papers lacked information about the device fingerprinting that is conducted using the API, which is the focus of this paper.

B. WebXR Emulator

The WebXR emulator also lacks a large volume of information, some of the source information about the tool was either deprecated or not entirely implemented. There is also a paper where the authors used the WebXR emulator, their study was mainly focused on bugs that are found within the WebXR API but because they used the same emulator it seemed like a useful idea to see what sort of issues they ran into. The emulator was an necessary part of our study because we were unable to obtain an actual AR/VR headset and had to make do with some emulation capabilities. [2] Unfortunately this paper was not of much help to find the bugs that were within the emulator extension itself, which meant that even if the findings were interesting they would not be of much use to us as we further studied fingerprinting with the emulator for WebXR.

C. WebXR Fingerprinting

There was not much out there in regards to fingerprinting with WebxR in terms of the literature that pertained to this specific subject. Instead we reviewed fingerprinting practices and did our best to understand exactly what was being identified by the WebXR API. In terms of fingerprinting we found that most of the fingerprinting is done from a Browser Specific fingerprinting and Cross Browser fingerprinting, though limited because of the extension capability and time-constraints there were ways that WebXR also uniquely identifies devices in other ways like canvas fingerprinting. To better understand the theory and bolster our understanding of Cross-Browser fingerprinting we turned to two papers in particular, both of which robustly define ways that cross-browser fingerprinting can be achieved [7], [8] WebXR functionality is determined by the type of device being used, these are determined by unique features that are included on each headset. In order ensure that the devices are handled per their specification needs the API needs to take fingerprints of the device that is connected. Browser-Specific fingerprinting is when the browser environment is used to gather more information about a user, in a study previously done researchers determined that a user could be identified by distinguishing whether or not a browser used Java or Flash[8] The other more relevant form of browser fingerprinting in the case of WebXR is Cross-Browser fingerprinting, which is independent of the browser this is typically done by obtaining an OS, screen resolution, system information, or sensors.[7], [8] In one study researchers were able to identify unique users by determining differences between user agent strings, this allowed them to figure out what type of OS was being used whether it was Windows, Linux, UNIX, MacOS, or some other OS.[7] In mobile browsers the nomenclature used for this type of fingerprinting is called Accelerometer fingerprinting, it is device specific measured values, and identifiers can be obtained without explicit privileges from a user.[8] Accelerometer fingerprinting is used by WebXR in the form of sensing whether or not a device has controllers and which or how the controllers are being used.[8]

D. WebXR Fingerprinting Mitigations and Defenses

Defenses against fingerprinting in regards to WebXR are another area where there is little to no literature. Fingerprinting that is being done on WebXR is still similar to the types that are already established. As such our team reviewed papers and literature that went over some current implementations for preventing fingerprinting from being achieved. W3 recommends browser implementations like this to prevent fingerprinting:

> "In order to improve consistency across specifications, the following requirements provide guidance for this balance:
>
> - Specifications MUST NOT increase the surface for passive fingerprinting unless a feature cannot reasonably be designed in another way.
> - Specifications SHOULD NOT increase the surface for active fingerprinting if comparable functionality could be accomplished without increasing the surface."

These suggestions however only relate to the browser itself, which begs the question how can users protect themselves from oversized fingerprinting surfaces? Well most fingerprinting is done in a passive and stateless manner which means most users, especially those who use WebXR enabled devices will be able to avoid or detect fingerprinting.[8] There are ways to spoof some fingerprints, or avoid them by using certain extensions, but at the moment the only way to mitigate fingerprinting is in the air and up for browser developers to decide upon.

VII. Conclusion

In this paper we explored the cross section of fingerprinting and the WebXR API. We aimed to produce meaningful fingerprinting results as well as evaluate effective defenses against this niche fingerprinting. We were able to produce a proof-of-concept web server that could collect unique fingerprints from a number of different VR and AR devices. We also were able to recommend several approaches that browser vendors and users could take to protect themselves from WebXR fingerprinting, even going so far as to build an extension that blocks WebXR.

This research shows a budding area of concern for security and privacy professionals. It surveys the intricacies of this

unique way to interact with digital spaces. Most importantly, we show a mature viewpoint of the WebXR API from an adversarial fingerprinting perspective.

References

[1] *Ublock origin - free, open-source ad content blocker.* uBlock Origin. [Online]. Available: https://ublockorigin.com/ (visited on 05/07/2022).

[2] *Augmented reality and virtual reality market size, share global forecast to 2025 — marketsandmarkets™*, www.marketsandmarkets.com. [Online]. Available: https://www.marketsandmarkets.com/Market-Reports/augmented-reality-virtual-reality-market-1185.html (visited on 05/07/2022).

[3] *What is fingerprinting and why you should block it*, en, https://www.mozilla.org/en-US/firefox/features/block-fingerprinting/.

[4] *Webxr device api - web apis — mdn$_2$022*, 2022. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API/.

[5] *What i learned from spending a week in virtual reality — jak wilmot — tedxvienna*, YouTube, Dec. 2019. [Online]. Available: https://www.youtube.com/watch?v=eX2QBlckPnw.

[6] N. Foundation, *Node.js*, Node.js, 2019. [Online]. Available: https://nodejs.org/en/.

[7] K. Boda, Á. Földes, G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," vol. 7161, Oct. 2011, ISBN: 978-3-642-29614-7. DOI: 10.1007/978-3-642-29615-4_4.

[8] R. Upathilake, Y. Li, and A. Matrawy, "A classification of web browser fingerprinting techniques," in *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, 2015, pp. 1–5. DOI: 10.1109/NTMS.2015.7266460.

[9] https://coveryourtracks.eff.org/..

[10] W3C, *What is fingerprinting? - w3*. [Online]. Available: https://www.w3.org/2014/strint/papers/41.pdf.

[11] *Webxr permissions and security - web apis — mdn*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API/Permissions_and_security#user_intent.

[12] *Xrsystem: Requestsession() - web apis — mdn*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/XRSystem/requestSession.

[13] *Content scripts - chrome developers*. [Online]. Available: https://developer.chrome.com/docs/extensions/mv3/content_scripts/#isolated_world.

[14] *Chrome.scripting - chrome developers*. [Online]. Available: https://developer.chrome.com/docs/extensions/reference/scripting/#method-executeScript.

[15] B. Maclntyre and T. F. Smith, "Thoughts on the future of webxr and the immersive web," in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2018, pp. 338–342. DOI: 10.1109/ISMAR-Adjunct.2018.00099.

[16] O. Renius, "A technical evaluation of the webxr device api for developing augmented reality web applications," M.S. thesis, Linköping University, Human-Centered systems, 2019, p. 28.

[17] U. Iqbal, S. Englehardt, and Z. Shafiq, "Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors," in *2021 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA: IEEE, May 2021.

[18] B. Deagon, *Facebook's apple-induced face plant could haunt company a while*, en, https://www.investors.com/news/technology/fb-stock-pummeled-by-apple-ad-changes/, Feb. 2022.

[19] B. Software, *Brave reaches 8 million monthly active users and delivers nearly 400 privacy-preserving ad campaigns*, en, https://brave.com/brave-reaches-8-million-monthly-active-users-and-delivers-nearly-400-privacy-preserving-ad-campaigns/, Oct. 2019.

[20] *The top browser fingerprinting techniques explained - FingerprintJS*, en, https://fingerprintjs.com/blog/browser-fingerprinting-techniques/..

[21] *WebXR API emulator – get this extension for firefox (en-US)*, en, https://addons.mozilla.org/en-US/firefox/addon/webxr-api-emulator/, Accessed: 2022-5-7, Aug. 2019.

[22] *All about browser fingerprinting and how to avoid this severe web threat*, en, https://privacysavvy.com/security/safe-browsing/browser-fingerprinting/., Jun. 2020.

[23] K. Boda, Á. Földes, G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," vol. 7161, Oct. 2011, ISBN: 978-3-642-29614-7. DOI: 10.1007/978-3-642-29615-4_4.

[24] *Webxr emulator extension*, GitHub, May 2022. [Online]. Available: https://github.com/MozillaReality/WebXR-emulator-extension (visited on 05/07/2022).

[25] *Webxr device api*, www.w3.org, Apr. 2022. [Online]. Available: https://www.w3.org/TR/webxr/.

[26] *Digital advertising spending in the u.s. 2023 — statista*, Statista, 2018. [Online]. Available: https://www.statista.com/statistics/242552/digital-advertising-spending-in-the-us/.

[27] S. Li, Y. Wu, Y. Liu, *et al.*, "An exploratory study of bugs in extended reality applications on the web," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, 2020, pp. 172–183. DOI: 10.1109/ISSRE5003.2020.00025.