ACCEPTED FROM OPEN CALL

REMOTE FINGERPRINTING OF MOBILE PHONES

VIMAL K. KHANNA

ABSTRACT

Mobile phone users access content from websites using their browsers and are tracked by cookies. Users also access content using mobile applications, which are tracked by totally separate identifiers, "device identifiers." Traditional approaches to tracking users fail to tie these two domains together. Hence, remote "fingerprinting" techniques have been suggested that extract set of attributes of a phone to create an almost unique identifier to track the phone user. We present a study of various fingerprinting techniques used at different layers of the networking protocol stack. We highlight the differences between fingerprinting computers and fingerprinting phones. Since fingerprinting is also a threat to user privacy, we present the current research and suggest future research directions in addressing the privacy issues.

INTRODUCTION

Mobile phone users use phone browsers to access web content from back-end content servers. They also run mobile applications (apps), which in turn access back-end content servers for content.

These content servers may need to uniquely track their users for various purposes, including protecting users from identity theft, preventing credit card fraud, serving advertisements relevant to them, and so on. For example, an e-commerce server can track the phone from which a user has initially registered his/her credit card information. Next time a user wants to make a purchase by entering the same credit card number, the server can check if the access is from the same phone, or it can ask for additional identifying information before approving the purchase.

Another use example can be that the server would like to display advertisements to the user. The server would like to keep track of contents accessed by the user across multiple sessions so that it can display relevant advertising to him/her on his/her next visit.

Hence, servers need to assign unique identifiers to each phone. In the mobile web domain, servers track users by storing unique cookies that persist across sessions. In the mobile apps domain, users are tracked by a totally different set of device identifiers. These different identifiers cause a server to wrongly interpret a single phone as two separate devices when the user accesses the web or runs an app.

Current research is focused on devising phone identification techniques that work across both web access and apps usage, that is, remote "fingerprinting" techniques.

Each phone has a set of hardware and software configurations. Fingerprinting techniques collect these configuration attributes from the phone for both web access and app usage. The attributes are sent to the back-end server, which concatenates them to create an almost unique "fingerprint" of the phone. Next time the same phone accesses the server, the server compares its fingerprint and determines if it is the same device. Hence, the user is effectively tracked over both the mobile web and app domains.

We present a study of various phone fingerprinting techniques being applied at different layers of the networking stack — the physical/data link, network, transport, and application layers. We show that although a number of the fingerprinting techniques in the online world can be applied to the mobile domain, some techniques fail due to the unique nature of mobile operator network infrastructure characteristics. Since Google's Android and Apple's iPhone iOS are the most prevalent smartphone operating systems, we mostly present examples from these platforms. A mobile phone is a personal device and is tied to a user. We use the terms "user/device/phone tracking" interchangeably in the text.

We start by describing typical mobile operator network architecture for mobile web access. We describe how cookies are used for tracking users. We then introduce the mobile apps domain and discuss how app usage is tracked by "device identifiers." We then show that these cookie- and device-identifier-based tracking approaches, in isolation, have limitations in uniquely tracking mobile phones. We then describe fingerprinting techniques that can work across both the mobile web and apps domains. We describe fingerprinting techniques used at various layers of the networking stack and identify a set of attributes to create almost unique fingerprints for phones. We discuss threats to user privacy due to fingerprinting and the current research on addressing the issue. We suggest future research directions for researchers/developers to comprehensively address the privacy issues.

MOBILE WEB

Users access websites using browsers on their phones. A user configures his/her phone to use the IP address of a mobile operator's Internet

Vimal K. Khanna is with mCalibre Technologies.

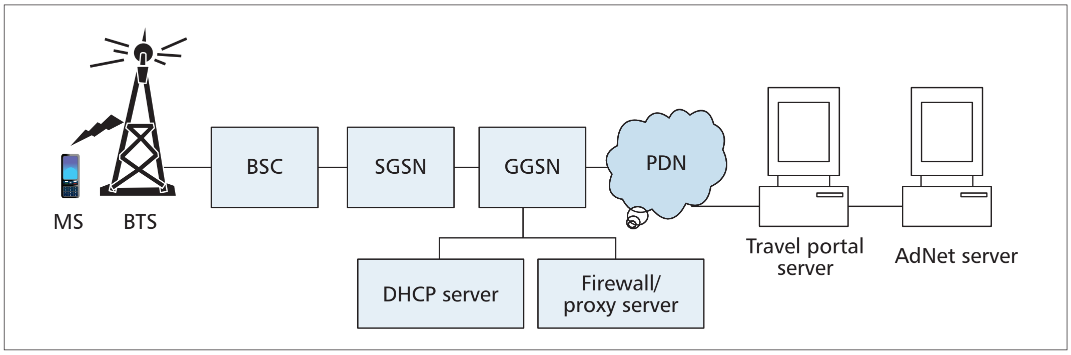

Figure 1. GPRS network.

A cookie is a small file sent from a website and stored on phone's browser directory. Let us assume that the Web Server is a Travel Portal. It stores its cookie and when user revisits the website, the browser sends its cookie back to the server. The server becomes aware that the phone is the same that accessed it earlier.

gateway. Each time he/she opens a website on his/her browser, a connection is made to this gateway, and through that the connection is completed to the website.

A technology used for making this connection is general packet radio service (GPRS), and its configuration is shown in Fig. 1 [1]. The phone, referred to as a mobile station (MS), connects to its nearest base transceiver station (BTS) over a wireless link. The BTS has an associated base station controller (BSC) that connects over a landline link to a mobile operator's Internet access infrastructure. The first component of this infrastructure is the serving GPRS support node (SGSN), which is primarily responsible for keeping track of the MSs it serves. The SGSN connects to a gateway GSN (GGSN), which interfaces with external packet data networks (PDNs). The web server to which the MS wants to connect is at the other end of the PDN.

Some GGSN components are shown in the figure. Each mobile operator is allocated a set of IP addresses to be allocated to its users. It has a Dynamic Host Configuration Protocol (DHCP) [2] component that allocates an IP address dynamically to the MS. The address is relevant only for the current session. Another GGSN component is a firewall/proxy server. The firewall protects the MS from attacks from the network. It blocks intrusive protocols from reaching the MS. Such protocols extract critical configuration information from the MS, which can be misused.

Similar functionalities are available in other types of mobile networks, such as edge, and third/fourth generation (3G/4G).

Once the user accesses the web server, the server tracks his device using cookies. A cookie is a small file sent from a website and stored on a phone's browser directory. Let us assume that the web server is a travel portal. It stores its cookie and when a user revisits the website, the browser sends its cookie back to the server. The server becomes aware that the phone is the same that accessed it earlier.

Furthermore, the travel portal may partner with an advertising network (AdNet). The content page of the travel portal has placeholders to display advertisements, which are served by the back-end AdNet server. AdNet stores its own third-party cookie on the browser to track a user's content access in order to serve relevant advertising.

Mobile Apps

A travel portal company can also build a TravelApp that users can download on their phones. If they need to book travel, they would run appropriate commands on the app, which will access the back-end travel portal to get them the right options.

AdNet will provide a software development kit (SDK) with libraries for displaying advertisements and tracking devices. The travel portal vendor links its TravelApp software with the SDK. The SDK library will request an advertisement from the AdNet portal and display it within the app.

The AdNet library also performs device identification by extracting a device identifier from the phone operating system (OS) and passing it back to the back-end AdNet portal.

In iPhones, prior to iOS 5, the app could extract an identifier called a "unique device identifier" (UDID), a string that uniquely identifies a specific iOS device.

However, some users do not like to be tracked by such identifiers. Options to disable UDID are absent. Hence, Apple deprecated UDID and introduced a new user-controlled identifier, "identifier for advertising" (IDFA). IDFA differs from UDID because a user can disable it. AdNet will then not be able to track this user.

The AdNet library could also extract the medium access control (MAC) address of a phone's WiFi card and also use it as an identifier. Apple has now deprecated extraction of MAC addresses too.

Similarly, for Android phones, the AdNet library can extract an identifier, "Android_ID," a string unique to each Android device. Furthermore, Android phones allow extraction of a complete set of other identifiers: MAC address, IMSI, and IMEI.

Since Android_ID suffers from the same shortcomings as Apple's deprecated UDID, Google decided to replace it with a new "Advertising ID" from Android 2.3 onward, including the current Android 5.0 (Lollipop) version. "Advertising ID" works like IDFA, in terms of the user being allowed to disable it.

Limitations of Traditional Tracking Approaches

The above tracking approaches have a major limitation in uniquely tracking mobile users. Advertising networks deliver advertisements to a user

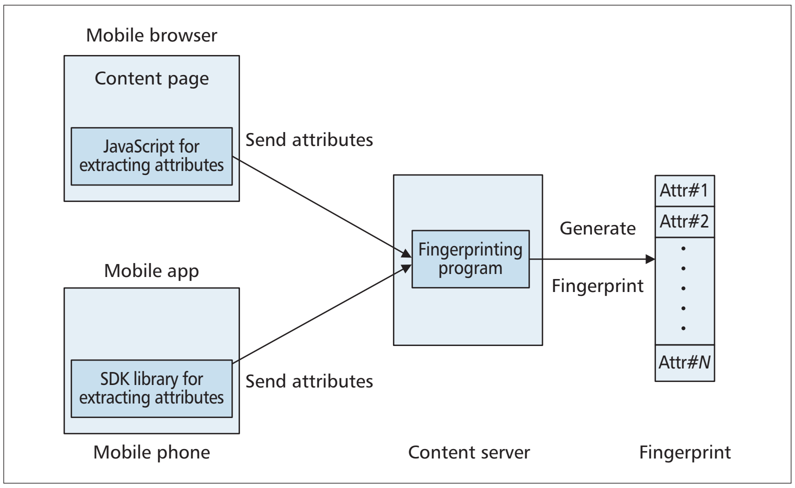

Figure 2. Fingerprinting a mobile phone.

while he/she is accessing content through both apps and the mobile web. As discussed, mobile web and apps are totally separate domains utilizing different identifiers: cookies and device identifiers, respectively. Due to these separate identifiers, *a single user is tracked as two separate individuals*, which is a big limitation for advertising networks wanting to track user behavior across both apps and web access (to serve advertisements of interest to him/her).

For example, a user could be a frequent traveller who always makes his travel bookings using TravelApp (and never on the travel portal website), and AdNet tracks his/her actions based on his device identifier. However, when the same user visits any website on the AdNet network, AdNet does not get his/her device identifier but only his/her cookie. Thus, AdNet is unaware that this user is quite interested in travel-related promotions. AdNet does not serve him/her a relevant travel promotion advertisement, leading to a lost business opportunity.

These limitations ask for a different approach to be followed for uniquely tracking a user, as discussed next.

Fingerprinting Techniques

Each phone has a number of hardware/software components: SIM card, hardware adaptors, clock, sensors, OS, installed apps/plugins, and so on. Each of these components is identified by its attributes; for example, the OS is identified by its name/version, plugins by their names/versions, and others. Each of these attributes in isolation may not be universally unique, but if a number of these are considered together, the device can be almost uniquely identified. Assimilation of such attributes into a single string comprises a device "fingerprint."

Fingerprints lose their accuracy over time as users keep updating their phones (e.g., loading a new app/plugin). Thus, commercial fingerprinting software products generally report the most likely match for a fingerprint as well as the accuracy of the match (e.g., 80 percent match). These products also recalculate the fingerprints at regular intervals so that fingerprints remain valid for longer times.

We now discuss techniques to be adopted and attributes to be extracted for phone fingerprinting at various layers of the networking stack — the physical/data link, networking, transport, and application layers. We show that a number of fingerprinting attributes used for tracking computers work in the mobile domain too. However, some of these attributes fail to work due to the unique nature of mobile operator networks.

A single fingerprint for a phone is built for web and apps domains so that a user can be tracked with the same fingerprint whether he/she is accessing the web through his/her browser or is using apps.

Browser: Let the user access the travel portal from his/her phone browser. AdNet adds an *invisible frame* in the content HTML page of the travel portal. This frame will have a JavaScript to extract various attributes from the phone browser. When the browser loads the content page, the frame is also loaded, and JavaScript is executed within the browser. The script extracts various phone/browser attributes and sends them to the back-end AdNet server. For example, it can extract phone time zone and installed plugins information. The back-end fingerprinting program on the AdNet server then collates all the collected attributes and creates a unique fingerprint for the device (Fig. 2). It compares the fingerprint with existing fingerprints in its database and declares the device to be the same as a previously seen device if a match is established. It can then send an advertisement to the browser based on past content access behavior of that user. If no match is found, it is considered a new device, and its fingerprint is stored for future matching.

Mobile App: TravelApp is linked to the AdNet SDK. SDK has a library that uses the system calls of the phone OS to extract various attributes. When a user runs the app, this library extracts and sends these attributes back to the same back-end fingerprinting program running on the AdNet server, as above (Fig. 2). The program assimilates all the attributes to create a fingerprint, compare it with existing fingerprints, and send an advertisement relevant to the user. The advertisement is then displayed within the app by AdNet's library on the phone.

We now discuss fingerprinting techniques used at different layers of the networking stack and various attributes that are collected.

Physical/Data Link Layer

Phones have a number of hardware and data link (MAC) layer components that can be attributes of a fingerprint: IMEI/IMSI/MAC address. As discussed earlier, some of these attributes can be extracted on Android and iOS.

Another hardware component used for tracking phones is its hardware clock. A clock consists of a crystal oscillator that ticks at a frequency, and a counter that counts total ticks to tell the "system time" of the phone. Clocks are built using inexpensive crystal oscillators, which are impacted by environmental factors and aging. Hence, a phone clock usually has some skew with respect to the global reference clock. It has been experimentally established that this skew is mostly unique on different phones and can be used as a fingerprint attribute.

The clock time of the phone can be gathered by a remote server by using various protocols at the network and transport layers (discussed in the next two sections). The time values are processed, and their difference from local time is calculated by the fingerprinting program on the server to determine the clock skew of the phone.

Smartphones also have a number of sensors (microphone, camera, GPS receiver, accelerometer, gyroscope, digital compass, etc.). Hardware imperfections can arise during the manufacturing process of such sensors. These imperfections can cause the sensor chips to behave in a non-ideal way. Such behavior patterns generate distinguishing features in each sensor chip, which allows it to be uniquely identified.

A number of mobile apps use the output of such sensors. Any such app can collect the output trace of a sensor and transfer it to a back-end fingerprinting server. The server can then extract distinguishing features of this output trace and build a fingerprint of the sensor, which would uniquely identify its phone device. We discuss such fingerprinting techniques later.

Network Layer

We now discuss the fingerprint attributes available from two network layer protocols: IP and Internet Control Message Protocol (ICMP).

IP Attributes: Each device accessing the Internet has an IP address [3]. An IP address could have been a potential attribute to permanently identify a phone. However, mobile operator networks assign IP addresses dynamically to a phone using DHCP (Fig. 1). A phone has different IP addresses for different network sessions. Hence, an IP address cannot uniquely track the phone across sessions.

However, an IP address still reveals some useful information about a phone and can be used to extract two fingerprint attributes:

- The IP subnet address reveals the "country" attribute of the phone user.
- Each mobile operator is assigned a pool of IP addresses that are globally known. Hence, an IP address is used to identify the "mobile operator" attribute of the user [4].

ICMP Attributes: ICMP [5] allows the server being accessed by a user to send various types of request packets to his/her phone. The ICMP module within the phone then sends appropriate response packets revealing parameters configured on the phone. Thus, the server fingerprinting program can actively exploit the ICMP protocol to fingerprint a phone by extracting phone attributes.

We discussed earlier that each phone generally has a different clock skew. The clock ticks are counted to calculate the system time of the phone. Differences in these clock skews result in system times of various phones being different from the global reference clock. ICMP defines a "timestamp" packet that carries the system time of the device.

The server issues ICMP Timestamp Request messages to the phone and records the trace of resulting Timestamp Reply messages. It compares server local system time at the instances the Timestamps Reply packets are received with these timestamps, and determines if the phone's clock is running faster or slower than its own clock. This difference is the "clock skew" of the phone compared to the server local clock, and it can be positive or negative. Clock skews are measured in parts per million (ppm), which is the clock difference between two devices in microseconds per second.

A plot of time difference between timestamps in received packets against local time on the server is used to determine a phone's clock skew (Fig. 3). The current server local time is plotted on the x-axis. The difference between local time and received timestamps in the packets is plotted on the y-axis for the complete packet trace. The slope of this plot is the clock skew, which is also the differential of the plot. Due to variable network delays, the timestamps will not always have fixed difference with local time, which results in the plot not being a straight line but rather a set of points, with upper and lower bounds. Linear programming techniques are then applied to determine the clock skew [6–8].

An analysis of ICMP timestamp-based fingerprinting of phones *over a WLAN* was performed for Android smartphones in [7]. Their experimental setup consisted of various smartphones (Samsung Google Nexus-S Android Jelly Bean 4.1.1, Motorola Motoluxe Android Gingerbread 2.3.6, Samsung Galaxy Mini S5570 Android Gingerbread 2.3.5, and Samsung I9000 Galaxy-S Android Gingerbread 2.3.6 phones) communicating with an access point using their WiFi MAC network adapters.

They observed that all these smartphones accept ICMP Timestamp Requests and respond to them. Furthermore, there is no option on the phone to block ICMP Timestamp Requests. Thus, these phones are amenable to ICMP-based fingerprinting.

They reported that differences in clock skews of these phones were in the range of tens of parts per million. Such differences are quite significant and clear, making clock skew an ideal fingerprint attribute. The reported precision of clock skew estimation was less than 1 ppm.

However, the phone being fingerprinted could be synchronizing its time with a global reference clock using Network Time Protocol (NTP) [9]. If a device continuously synchronizes its local time with the global time from an NTP server, its ICMP timestamps will carry global time. These timestamp values would then be independent of the frequency of the clock of the device, and the fingerprinting server would not be able to determine the clock skew.

Experiments conducted in [10] showed that if NTP is enabled on computers running Linux post-2007 versions, clock skew cannot be determined by fingerprinting servers. Although they did not conduct experiments for smartphone operating systems, they argued that since Android OS is based on Linux, similar behavior could be observed for smartphones too.

However, the analysis of Android phones in [7] shows that although ICMP timestamp values change due to NTP synchronization, there are still enough time gaps between NTP synchroni-

The ICMP protocol module within the phone then sends appropriate response packets revealing parameters configured on the phone. Thus, the server fingerprinting program can actively exploit the ICMP protocol to fingerprint a phone by extracting phone attributes.

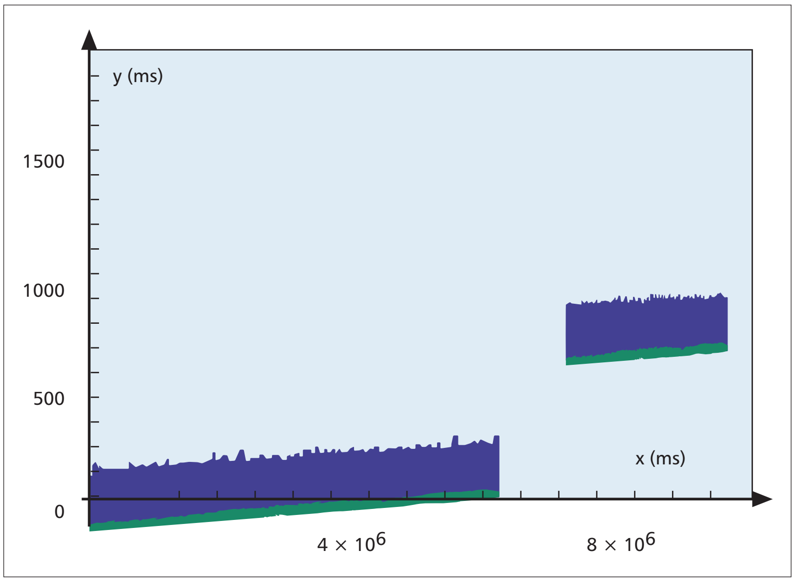

Figure 3. Clock skew estimation, NTP-enabled (figure reproduced from [7]).

zation instances to allow the fingerprinting server to estimate the clock skew.

Each time the fingerprinted phone performs NTP time synchronization, the Timestamp field in the ICMP Response packet changes, and a discontinuity appears in the plot of its packet trace timestamps. However, the *slope of the plot* before and after this synchronization instance still remains the same (Fig. 3). The clock skew is measured by ignoring this discontinuous instance. The clock skew thus estimated is stable despite such time synchronizations [7].

The reason for Android phones behaving differently from Linux computers is because the NTP synchronizations on phones are generally performed at a slower rate than on computers. Phones are battery-powered devices and need to conserve energy. Since NTP synchronizations have processing overheads that consume power, phones are made to synchronize less frequently with the NTP servers. Furthermore, mobile operators charge subscribers for their data usage. The subscribers control their costs by reducing their data transfers by limiting the frequency of NTP synchronizations. The low frequency of NTP synchronizations allows the fingerprinting servers to estimate the clock skew of the phones.

From the above discussion, it might seem that ICMP is an effective method to estimate clock skews. However, the experiment in [7] was performed over a WLAN environment, without the phone having to go through a mobile operator network before accessing the server. Mobile operator networks can have firewalls associated with the GGSN node (Fig. 1). ICMP messages are now blocked by many firewalls to protect phone users. In such scenarios, the ICMP Timestamp Request packets does not receive any responses from the phone, and the fingerprinting program is unable to determine the clock skew.

Due to this limitation, ICMP is not the right option to determine clock skews of phones, and an alternative approach needs to be followed. We now discuss how a transport-layer-based approach can be used to determine clock skew.

Transport Layer

TCP stacks over smartphones implement the TCP timestamps option defined in Internet Engineering Task Force (IETF) RFC 1323 [11]. Each party in a TCP flow includes information about its perception of time in the "Options" field of outgoing Data/ACK packets. The option carries the current Timestamp value of the phone.

The RFC states that the timestamps should be taken from a "virtual clock" that is "at least approximately proportional to real time." The Timestamp value is incremented by a counter. The counter is incremented after a predefined number of ticks of the local hardware clock. Hence, timestamps in subsequent packets will have values dependent on the frequency of the local clock hardware. Thus, there is a direct relation between TCP timestamps and the phone's local clock frequency, which can be exploited by the remote fingerprinting server to estimate the phone's clock skew.

The method of determining the clock skew is similar to the one described in the last section. The phone accesses the content on the server. The fingerprinting program on the server captures the trace of timestamps in the TCP packets received during the content access. The time difference between these timestamps and server local time is plotted, and linear programming techniques are used to determine the clock skew from the slope of the plot [6, 8].

Since these timestamps are carried in TCP Data/ACK packets themselves, they are *not blocked by firewalls* in the mobile operator network.

A detailed work on estimating clock skews for phones has shown that TCP timestamps are quite effective for the purpose [8]. They have measured clock skews of Samsung Galaxy GT-9000 Android 2.1 and Nokia N8 Symbian 3 phones. They have observed that the TCP timestamp option is enabled by default on these OSs, making these phones amenable to fingerprinting. Although they used Android v. 2.1, these observations are valid for the latest versions of Android too, since the TCP timestamp option is also enabled by default on Android 5.0.

Their work reported that each phone had a unique clock skew, and there was no visible impact on skew estimation even with NTP synchronization. Their results show that the skew estimates remained stable and did not fluctuate by more than 0.1–0.3 ppm when the measurements were repeated even after a few months.

It should be noted that to plot the timestamps of a device, the fingerprinting server should first be able to uniquely identify the packets coming from that device and distinguish them from packets coming from other devices. A device is usually identified by its IP address for the duration of its current session. Each device has a unique IP address for a session (even if dynamically allocated for that session). The server plots the timestamps of the packets coming from an IP address and determines the clock skew of the device corresponding to that address. It then maps that device to a device seen previously with the same clock skew even if that device had a different IP address then.

One might assume that if the IP address itself is obfuscated by assigning the same IP address to multiple devices, it might be possible to prevent the estimation of devices' clock skew.

One protocol used for this purpose is Network Address Translation (NAT) [12]. A network with multiple devices can have a NAT server connecting them to the global Internet. NAT has a single outgoing global IP address that is used for all its devices.

When a device starts a TCP connection to a remote server, it sends a TCP packet containing its (Local Source IP Address, Source TCP Port) values to the NAT server. NAT assigns a new unique TCP port to this connection. It maps the (Local Source IP Address, Source TCP Port) tuple to the (Global IP Address, Assigned TCP Port) tuple and replaces these values in the outgoing packets.

Thus, all the devices communicating with the fingerprinting server will carry the same IP address. When the server sends a response packet to a device, NAT changes the incoming (Global IP Address, Assigned TCP Port) to the mapped (Local Source IP Address, Source TCP Port) values, and delivers the packet to the actual local device.

However, one limitation of NAT is that while it performs address/port translations, it does not rewrite the TCP timestamp option in the outgoing packets.

Multiple techniques have been suggested that exploit this loophole and use TCP timestamps to uniquely identify devices, despite their sharing the same global IP address [6, 13–16]. Once the devices have been uniquely identified, their clock skew can then be estimated.

The techniques involve plotting the TCP timestamp values of packets being received from an IP address against the local time of the server. When a device boots, its initial timestamp value is set to 0 in case of Linux, or to a random number for some other OSs. Since the devices would boot at different times, their plots of timestamps would show measurably different offsets.

Thus, one observes multiple clusters of points that are at a distance from each other. Each cluster of points corresponds to a different device. These individual clusters are converted to lines using the least squares fit method. The clock skews of the corresponding devices are then determined using linear programming techniques, as discussed earlier.

It is possible that some devices may boot at the same time instance. In such a case the timestamp plots of these devices would show intersecting points. However, since these devices would have different clock skews, one would start observing multiple lines emerging from these intersection points as more time elapses. Each line would have a different slope, which can be used to distinguish these devices and measure their clock skews.

Thus, TCP-timestamp-based fingerprinting methods can even defeat the privacy protection provided by NATs. A possible solution to prevent this possibility is by making NAT rewrite the timestamps in outgoing TCP packets, which is currently not being done.

Application Layer

A number of fingerprinting attributes can be collected at the application layer. Phone users access websites from their browsers using HTTP. HTTP protocol headers carry information about the device/browser configuration. This information can be collected by the fingerprinting program at the server to be used as fingerprint attributes [17, 18]. Some relevant HTTP headers and their contents are:

User Agent (UA):	Browser name/version, OS name/version, phone model, language
Accept:	Data types supported
Accept-Encoding:	Browser encoding capabilities
Accept-Charset:	Allowable character sets

Furthermore, the fingerprinting server embeds a JavaScript on the webpage being accessed, which extracts additional information, such as time zone, screen resolution, plugins installed, and information extracted through them, if cookies are enabled, and if Java is supported.

Using Mobile Apps for Fingerprinting Sensors

As discussed earlier, smartphones have a number of sensors with distinguishing characteristics that can be used to fingerprint them. Since sensor data from phones is extracted by mobile apps (and sent to back-end servers), we consider these mechanisms as belonging to application layer fingerprinting techniques.

One such example of fingerprinting sensors is discussed in [19], where they have fingerprinted Accelerometer sensors in phones. An accelerometer sensor detects and measures the tilt, motion, rotation, swinging, and shaking of a phone. The target applications of accelerometer sensors are gesture recognition, display rotation, motion-enabled games, fitness monitoring, and so on.

Hardware imperfections during the sensor manufacturing process can cause different accelerometer sensor chips to respond differently to the same motion stimulus. Thus, the output data of the accelerometer sensor of one phone would be different from those of other phones, allowing the phone to be fingerprinted.

Users download mobile apps on their phones that use accelerometer functions, say, a motion-enabled gaming app. The app uses the output data of the accelerometer sensor to perform its functions. If the app also decides to send this data to a back-end fingerprinting server, the fingerprinting application can build a fingerprint of the device from this output trace.

The experiment in [19] involves sending the accelerometer output from the phone to a back-end server that plots its complete output trace. They observed that when the phone vibrates, its accelerometer trace is distinguishably different from the trace during normal human activity. From the output data trace they identified and extracted portions corresponding to such vibration periods for each phone.

They used a feature extraction software tool to extract a vector of 36 features from each indi-

Smartphones have a number of sensors with distinguishing characteristics that can be used to fingerprint them. Since sensor data from phones is extracted by mobile apps (and sent to backend servers), we consider these mechanisms as belonging to application layer fingerprinting techniques.

A method to prevent exact fingerprint creation is to forge the attributes sent in UA string by browsers. However, fingerprinting programs collect a number of other attributes and can detect a mismatch, like the UA reporting "iPhone" but a Flash Player plugin is seen installed (iPhone does not support Flash).

vidual trace. They ran classification techniques on these features to generate a unique fingerprint for each phone corresponding to a trace.

They ran the experiment on 25 Android phones of six different models: Nexus One, Samsung Galaxy Nexus, Samsung Galaxy S3, Nexus S, HTC Incredible Two, and HTC MyTouch. Their results show that each phone has a unique fingerprint, with precision upward of 96 percent.

Privacy Issues and Future Research Directions

Fingerprinting techniques are a serious threat to the privacy of users, even more grave than threats imposed by cookies. Since cookies are "physical files" placed on the phone, users can configure their browsers to block them or periodically delete them. In contrast, fingerprinting techniques just extract information from phones and leave no persistent evidence. There are no traces to be blocked/removed, and a different set of approaches need to be adopted to prevent a user from being fingerprinted.

We now discuss the current research being conducted to prevent fingerprinting techniques from invading user privacy. Since some issues are still unresolved in this field, we suggest future research directions that researchers/developers can follow to provide comprehensive solutions to address this problem.

Preventing Clock Skew Estimation

Fingerprinting servers estimate the clock skew from TCP Timestamp Option field values in the packets received from the phone. Hence, the phone can be protected by disabling the TCP Timestamp Option in outgoing packets. However, this approach is undesirable because the use of TCP timestamps improves round-trip time (RTT) estimation and TCP performance.

The OpenBSD OS on computers offers some solutions that modify TCP timestamps to prevent detection of clock skew.

Reassemble TCP Option: Clock skew can be estimated from TCP timestamps because their values are dependent on the clock tick value of the device. If, instead, these timestamps are made monotonically increasing and not derived off a guessable base time, they do not reveal the clock skew information. OpenBSD provides a "Reassemble TCP" option that modifies the TCP timestamps in outgoing packets on the device with random and monotonically increasing numbers [20].

TCP Connection-Specific Timestamps: The fingerprinting server distinguishes devices behind a NAT by using their TCP timestamp values. An option is provided in OpenBSD to allow a device to use a separate TCP timestamp clock for each active TCP connection [14]. The fingerprinting server would then observe multiple TCP timestamp traces for a single device. Thus, it would not be able to uniquely identify the device, which would prevent estimating the clock skew.

Researchers should implement these OpenBSD options in TCP stacks of various smartphone OSs to defeat clock-skew-based fingerprinting possibilities.

Researchers should also note that even if TCP timestamp values are obfuscated, clock skew can still be estimated using some other methods. It is observed that some systems send packets at specific time intervals, perhaps due to interrupt processing times. The fingerprinting server can apply Fourier transform on packet arrival times to estimate the frequency at which a phone transmits packets. The clock skew so estimated has been reported to be close to that estimated using TCP timestamps [6]. Researchers should work on finding a solution to this problem.

Another suggested area of research is to find ways to reduce phone clock skews by implementing more precise "software clocks" in smartphones by basing the clock on CPU cycles, as has been done on Linux PCs [21].

Fingerprinting-Resistant Browsers

Since a number of application layer fingerprinting attributes are extracted by Javascript, one can prevent it by disabling Javascript in the browser. However, this option significantly degrades the browsing experience.

A method to prevent exact fingerprint creation is to forge the attributes sent in UA strings by browsers. However, fingerprinting programs collect a number of other attributes and can detect a mismatch, like a UA reporting "iPhone" but a Flash Player plugin being seen installed (iPhone does not support Flash).

Much of the differentiation in fingerprint attributes is because of precise micro-version numbers of attributes received in UA strings and in the names of plugins. Hence, browser/plugin developers can decide to make version numbers less precise (say, "DivX 1.4" rather than "DivX 1.4.0.233"). However, developers need such micro-versioning for *debuggability* — to diagnose reported errors in a particular micro-version of their code. Developers should work on devising means to strike a balance between debuggability and privacy.

Researchers have developed the Tor Browser [22] with multiple features that prevent reporting phone hardware/software configuration to fingerprinting programs:

- Identical UA/HTTP header set/time zone is reported for all devices.
- All plugins are disabled (can be activated only by explicit clicking by user).
- Total device screen width and height are reported to be same as the available screen width and height (inner content window), preventing a fingerprinting program to decipher the actual screen size of the device.

However, it is a known paradox that some anti-fingerprinting techniques can instead make a device more uniquely identifiable (fingerpintable) unless the techniques are widely used. For example, usually the available screen height is less than total device screen height due to the taskbar. If a fingerprinting program notices that both heights are the same, it can easily interpret that the device has the Tor Browser. Hence, an additional fingerprinting attribute becomes available, which makes Tor Browser users easily identifiable if only a very small number of them are accessing that server.

We suggest that researchers further scrutinize and contribute to the Tor Browser project to develop more robust privacy-enhancing browser techniques.

Preventing Sensors' Fingerprinting

As discussed earlier, distinguishing characteristics of sensors in phones can allow them to be fingerprinted. Researchers should look at all types of sensors available in smartphones and explore all possible threats by which they can be fingerprinted.

The output traces of these sensors are extracted by mobile apps. Mechanisms used by these apps to access and use such sensor data would broadly be similar across various types of sensors. Thus, researchers should first develop generic policy frameworks that would allow phone users to precisely control the access rights of mobile apps over sensor data, limit the sensor data that apps are allowed to transfer to back-end servers, and so on.

Individual-level solutions to prevent fingerprinting of each type of sensor should then be developed by considering the characteristics of individual sensors. For example, we earlier discussed the case of a motion-enabled gaming app sending the accelerometer sensor data to a back-end server, which generates a fingerprint due to unique characteristics displayed by each accelerometer sensor. However, we are aware that a gaming app needs the accelerometer data only for its local use, to run relevant features of the game based on the hand movement of the user. The gaming app does not need to send this data to a back-end server for any purpose. Thus, researchers should implement policy mechanisms for mobile apps that would allow a user to block the gaming app from sending the accelerometer sensor data to a back-end server. Once the user enforces this policy on his/her device, the back-end server of the gaming app would not be able to fingerprint the phone.

Researchers should implement policy mechanisms for mobile apps that would allow a user to block the gaming app from sending the accelerometer sensor data to a backend server. Once the user enforces this policy on his device then the backend server of the gaming app would not be able to fingerprint the phone.

Conclusion

We have presented the problems in uniquely tracking mobile phone users across websites and mobile apps usage. We have described fingerprinting techniques that extract a set of attributes of a phone to create an almost unique identifier to track the phone user across web and apps usage. We have identified various fingerprinting attributes, such as MAC layer identifiers, clock skew, sensor characteristics, IP address, OS name/version, browser characteristics, and plug-in characteristics. Techniques used to extract these attributes at different layers of the networking protocol stack have then been presented. We have shown that some techniques being used for fingerprinting computers, like ICMP based clock skew estimation, fail to work for fingerprinting phones due to such protocols being blocked by firewalls in mobile operator networks. We have discussed how fingerprinting programs on web servers use HTTP information and embedded Javascripts on their web pages to extract a range of attributes from phones and generate unique fingerprints. We have also pointed out that mobile apps can send local sensor data to back-end servers to generate fingerprints. We have then presented the current research in protecting user privacy by preventing fingerprinting of phones — modifying TCP timestamps to prevent clock skew detection, using browsers that avoid reporting phone attributes to back-end fingerprinting programs, and so on. Since some issues still remain unresolved in this research domain, we have suggested future research directions that researchers/developers can follow to provide comprehensive solutions to prevent fingerprinting of phones.

References

[1] B. Ghribi and L. Logrippo, "Understanding GPRS: The GSM Packet Radio Service," *Computer Networks 34*, 2000.

[2] R. Droms, "Dynamic Host Configuration Protocol," IETF RFC 2131.

[3] J. Postel, "Internet Protocol," IETF RFC 791.

[4] I. Krontiris, F. Freiling, and T. Dimitriou, "Location Privacy in Urban Sensing Networks: Research Challenges and Directions," *IEEE Wireless Commun*., vol. 17, no. 5, Oct. 2010.

[5] J. Postel, "Internet Control Message Protocol," IETF RFC 792.

[6] T. Kohno, A. Broido, and K. Claffy, "Remote Physical Device Fingerprinting," *IEEE Trans. Dependable Secure Comp.*, vol. 2, no. 2, Apr.–June 2005.

[7] M. Cristea and B. Groza, "Fingerprinting Smartphones Remotely via ICMP Timestamps," *IEEE Commun. Lett*., vol. 17, no. 6, June 2013.

[8] S. Sharma, H. Saran, and S. Bansal, "An Empirical Study of Clock Skew Behavior in Modern Mobile and Hand-Held Devices," *Proc. Third Int'l. Conf. Commun. Systems and Networks*, 2011.

[9] D. Mills, "Network Time Protocol," IETF RFC 5905.

[10] L. Polcak, J. Jirasek, and P. Matousek "Comment on Remote Physical Device Fingerprinting," *IEEE Trans. Dependable Secure Comp*., vol. 11, no. 5, Sept./Oct. 2014.

[11] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," IETF RFC 1323.

[12] P. Srisuresh, "Traditional IP Network Address Translator (Traditional NAT)," IETF RFC 3022.

[13] A. Tekeoglu, N. Altiparmak, and A. Tosun, "Approximating the Number of Active Nodes behind a NAT Device," *Proc. 20th Int'l. Conf. Comp. Commun. and Networks*, 2011; http://cecs.louisville.edu/nihat/papers/ICCCN2011.pdf

[14] G. Wicherski, F. Weingarten, and U. Meyer, "IP Agnostic Real-Time Traffic Filtering and Host Identification Using TCP Timestamps," *Proc. IEEE 38th Conf. Local Comp. Networks*, 2013. https://itsec.rwth-aachen.de/publications/natfilterd_paper.pdf

[15] E. Bursztein, "TCP Timestampto Count Hosts behind NAT," *Phrack Mag.*, issue 63, article 0x03-2, Aug. 2005; http://phrack.org/issues/63/3.html

[16] E. Bursztein, "Time Has Something to Tell Us about Network Address Translation," 12th Nordic Wksp. Secure IT Sys., 2007; http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/Bur-nordsec07.pdf

[17] P. Eckersley, "How Unique Is Your Web Browser?" *Proc. 10th Int'l. Conf. Privacy Enhancing Technologies*, 2010.

[18] R. Broenink, "Using Browser Properties for Fingerprinting Purposes," *Proc. 16th Twente Student Conf. IT*, 2012.

[19] S. Dey *et al*., "AccelPrint: Imperfections of Accelerometers Make Smartphones Trackable," *Network and Distrib. Sys. Security Symp.*, 2014; http://synrg.csl.illinois.edu/papers/AccelPrint_NDSS14.pdf

[20] Jose, "Reassemble TCP," *OpenBSD J.*, Oct. 2003, http://undeadly.org/cgi?action=article&sid=20031030075038

[21] A. Pasztor and D. Veitch, "PC Based Precision Timing without GPS," *Proc. SIGMETRICS Conf.*, 2002.

[22] M. Perry, E. Clark, and S. Murdoch, "The Design and Implementation of the Tor Browser," Mar. 2013; https://www.torproject.org/projects/torbrowser/design/.

Biography

Vimal K. Khanna is founder and managing director of mCalibre Technologies. He has 30 years of experience in the software industry. He is a Technical Editor of *IEEE Communications Magazine* and is also listed in *Marquis Who's Who in the World*. His sole-computer networking and operating systems technical papers have been published in IEEE/ACM journals. His work has been cited on the Linux STREAMS Open Source Community website. He has presented numerous project management papers at PMI Global Congresses.