VB2020 localhost

30 September - 2 October, 2020 / vblocalhost.com

HIDDEN RISKS OF ADVERTISEMENTS

Doina Cosovan & Catalin Valeriu Lita

Security Scorecard, Romania

dcosovan@securityscorecard.io
cvaleriu@securityscorecard.io

ABSTRACT

Advertising is everywhere. Mobile applications are widely used and most of them, especially the free ones, embed advertising software development kits (SDKs). Web browsing frequently exposes users to advertisements, as well.

There are multiple methods for sinkholing advertising infrastructure. First, advertisers may let their hard-coded domains expire. Second, they can use wrong domains, either intentionally or unintentionally. Third, some advertising SDKs started to use domain generation algorithms (DGAs) as a fallback mechanism for the hard-coded domains. This seems to be an attempt to bypass ad blockers, which could have blacklisted their hard-coded domains. In the last case, using an ad blocker turns out to be more dangerous than not using one as it exposes the user to sinkholable domains, which might be under an attacker's control.

This paper focuses on analysing the security risks involved when an advertising infrastructure can be sinkholed. First, an attacker can passively gather, for later use, personally identifiable information about both the users and the advertisers. Second, an attacker can actively serve specially crafted advertisements, including malvertising, in order to exploit and infect the contacting systems.

We have sinkholed multiple advertising-related domains, but we will illustrate a few interesting use cases.

The first use case is an *Android* mobile advertising SDK with an expired hard-coded domain, contacted by more than half a million unique IP addresses daily from tens of different applications.

The second use case is a multi-platform mobile advertising SDK using a hard-coded domain, which receives requests from more than 10 million unique IP addresses per day.

The third use case is an advertising platform for browsers, whose domain was inserted in the SDK by the developers with a typo. It was contacted by more than two million unique IP addresses daily and was used by more than 10,000 different websites until the problem was fixed a week later.

The fourth use case is a browser HTML5 advertising player that intentionally used a non-resolving domain in order to evade syntax errors. They fixed the issue a few days after we sinkholed the domain, but during those few days, the sinkhole received requests from half a million unique IP addresses.

The fifth use case is a multi-platform mobile advertising mediator with a fallback DGA, which is used in more than 1,000 different applications and receives requests from tens of thousands of unique IP addresses (note that this is only the subset of IP addresses that didn't manage to contact the hard-coded domain out of the total number of IP addresses using the advertising mediator).

For one of the presented use cases, we created a proof of concept illustrating how an attacker can serve a specially crafted advertisement by owning an infrastructure domain. The most concerning aspects are: this happens regularly, this happens for the latest official advertising platforms / SDKs, and this happens for advertisers that have a large user base.

1. INTRODUCTION

Some advertisers are negligent when it comes to infrastructure domains. This allowed us to sinkhole multiple advertising-related domains. In this paper we will illustrate a few interesting real-world use cases.

Furthermore, this paper focuses on what the attackers might be able to achieve in various real situations we stumbled upon.

2. USE CASES FOR EACH SITUATION

In this section we will illustrate at least one use case for each of the following three situations, that made sinkholing advertising infrastructure possible:

1. Expired hard-coded domains
2. Usage of wrong domains
3. Available algorithmically generated domains.

2.1 Expired hard-coded domains

It is not unheard of to discover expired domains which are associated with servers that are still being contacted by various clients [1]. Depending on the use case, these domains present various types of risks and can be of great interest for attackers. Advertisers are letting infrastructure-related hard-coded domains expire, too.

Of all the existing situations that make sinkholing of advertising-related domains possible, this is by far the most prevalent. Therefore, for this situation, we present two different use cases.

2.1.1 Use case 1: Android game applications using an advertising SDK

The first use case features an *Android* mobile advertising SDK, which contacts an expired hard-coded domain. After sinkholing it, this domain was contacted by more than half a million unique IP addresses daily.

It receives different types of requests. These are a few examples:

```
1. GET /m/gdpr_sync?id=[redacted]&nv=5.5.0%2Bunity&current_consent_status=unknown&force_gdpr_
applies=0&bundle=[redacted]&dnt=0 HTTP/1.1
User-Agent: Mozilla/5.0 (Linux; Android 7.0; SM-G935F Build/NRD90M; wv) AppleWebKit/537.36
(KHTML, like Gecko) Version/4.0 Chrome/79.0.3945.136 Mobile Safari/537.36
Accept-Language: pt-br
Host: [redacted]
Connection: Keep-Alive
Accept-Encoding: gzip

2. POST /m/gdpr_sync HTTP/1.1
Accept-Language: th-th
User-Agent: Dalvik/2.1.0 (Linux; U; Android 9; FLA-LX2 Build/HUAWEIFLA-LX2)
Content-Type: application/json; charset=UTF-8
Host: [redacted]
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 172

{
       "current_consent_status":"unknown",
       "nv":"5.5.0+unity",
       "force_gdpr_applies":"0",
       "id":"[redacted]",
       "dnt":"0",
       "bundle":"[redacted]"
}

3. POST /m/open HTTP/1.1
Accept-Language: en-us
User-Agent: Mozilla/5.0 (Linux; Android 8.1.0; LM-Q710.FGN Build/OPM1.171019.019; wv)
AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/79.0.3945.136 Mobile Safari/537.36
Content-Type: application/json; charset=UTF-8
Host: [redacted]
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 218

{
       "st":"1",
       "current_consent_status":"unknown",
       "av":"0.5.1",
       "v":"6",
       "nv":"5.5.0+unity",
       "force_gdpr_applies":"0",
       "dnt":"0",
       "id":"[redacted]",
       "udid":"[redacted]"
}
```

Each request has a field called ‘bundle’, which contains the name of the package for the application from which the request was sent. Therefore we were able to compute some statistics about the applications that are using this particular SDK: approximately 40 different applications, most of which are related to gaming. Of the top 10 of these applications, sorted by the number of requests received from them, two have multiple millions of installs and another three have around one million installs.

2.1.2 Use case 2: Advertising SDK used by websites and mobile applications related to social media and news

Another use case for this situation is a multi-platform advertising SDK which is contacting an expired hard-coded domain. By sinkholing the domain, we received requests from more than 10 million unique IP addresses daily. Almost 70% of those IP addresses were located in the United States.

The following are examples of received requests:

```
1. GET /sync/dsp?uid=[redacted]&partner=[redacted]&zone=[redacted] HTTP/1.1
Host: [redacted]
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/83.0.4103.61 Safari/537.36
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Sec-Fetch-Dest: image
Referer: [redacted]
Accept-Encoding: gzip, deflate, br
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8

2. GET /sync/dsp?uid=[redacted]&partner=[redacted]&zone=[redacted] HTTP/1.1
Host: [redacted]
Connection: keep-alive
User-Agent: Mozilla/5.0 (Linux; Android 8.0.0; SM-G930V Build/R16NW; wv) AppleWebKit/537.36
(KHTML, like Gecko) Version/4.0 Chrome/75.0.3770.67 Mobile Safari/537.36 [FB_IAB/
FB4A;FBAV/245.0.0.39.118;]
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Referer: [redacted]
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
X-Requested-With: [redacted]
```

The HTTP Referer header contains the URL from which the request was made. This allowed us to figure out which websites are using this platform. We found that it is used by 10,000+ different websites, most of which seem to be related to social media and news. Out of the top 10 referrer domains sorted by the number of received requests, two are ranked in the top 100 while five are ranked in the top 1,000 in global Internet engagement according to *Alexa*.

The HTTP X-Requested-With header contains the name of the application's package. This allowed us to figure out which mobile applications are using this platform. It seems to be used by almost 2,000 different applications, most of which are also related to social media and news. Of the top 15 applications sorted by the number of requests received from them, eight have millions of installs and another three have around 1 million installs.

2.2 Usage of wrong domains

Although this seems surprising, we found both intentional and unintentional uses of wrong domains in advertising. We present one use case for each of them.

2.2.1 Use case 3: unintentional use of wrong domain by a browser and mobile advertising platform

A developer of an advertising platform, just as any person, is prone to making mistakes. In this section, we discuss a use case in which a wrong domain was hard coded in an advertising platform.

Specifically, the domain was inserted in the SDK with a typo: the domain is missing an 'a' character. We know this because the advertisers bought the correct domain, but distributed the code with the wrong domain. We wanted to contact them, but we couldn't find a way to do so because the Whois information was protected. Regardless, they figured out their mistake in a little over a week. We sinkholed the domain with the typo on 14 March 2020 and they fixed the typo on 24 March 2020.

During the short period of time in which the code used the domain under our control (the domain with the typo), it received approximately 36 million requests from approximately 2.5 million unique IP addresses daily. More than 90% of these IP addresses were located in Russia, as would have been expected given the fact that the hard-coded domain name uses the '.ru' top level domain.

This is an example of a received request:

```
GET /userbind?src=[redacted]&pbf=1&gi=1 HTTP/1.1
Host: [redacted]
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/80.0.3987.149 Safari/537.36
Sec-Fetch-Dest: image
Accept: image/webp,image/apng,image/*,*/*;q=0.8
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: no-cors
Referer: [redacted]
```

```
Accept-Encoding: gzip, deflate, br
Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7
```

This advertising platform seems to be used for both browsers and mobile applications. The HTTP Referer header allowed us to figure out which websites are using the platform. We found that it is used by 20,000+ different websites, most of which seem to be related to gaming and movies.

The HTTP X-Requested-With header allowed us to figure out which mobile applications are using this platform. It seems to be used by almost 1,000 different applications, most of which are related to social networking and browsing. Of the top 15 applications sorted by the number of requests received from them, 13 have millions of installs.

2.2.2 Use case 4: intentional use of wrong domain by an HTML5 advertising player

Although this sounds counterintuitive, we stumbled upon a case in which an advertiser used a wrong domain intentionally.

This use case features a browser HTML5 advertising player that was intentionally using a non-resolving domain in order to evade syntax errors.

This is how the code looked:

```
if (xam == 1) {
 var xameleon = 'https://ssp.xameleon.io/?[redacted]&page='+referer+' and ';
} else {
 var xameleon = '';
}

if (buz == 1) {
 var buzzoola = 'https://exchange.buzzoola.com/adv/[redacted]/jsvpaid';
} else {
 var buzzoola = '';
}

var player = new Playerjs({
 id:"player",
 file:"https://www.youtube.com/embed/"+id_arr_1.rand(),
 …,
 preroll: "" + imhop + "" + nativimho + "" + instream + "" + instinread + "" + instream568 + ""
+ independant + "" + independant2 + "" + adspgpmd + "" + nativ + "" + instream + "" + smby + ""
+ otm1 + "" + imho + "" + nativ2 + "" + yandex + "" + otm3 + "" + advarkads2 + "" + adriver +
"" + advarkads + "" + vihub1 + "" + vihub2 + "" + vihub3 + "" + adspektr + "" + skwstat + "" +
betweendigital + "" + republer + "" + adspmt + "" + utraff + "" + otm2 + "" + kinoaction + "" +
kinoaction2 + "" + kinoaction3 + "" + videonow + "" + vhead + "" + xameleon + "" + buzzoola +
"https://321[redacted].ru"
});
```

Note the preroll field of the player which is initialized with a concatenation of various variables. The correctness of the syntax for the resulting *preroll* variable depends on whether the variables *xameleon* and *buzzoola* are empty or not. For example, if we consider only those two variabiles, there are four possible cases:

1. Both variables are initialized
2. Both variables are empty
3. The variable *xameleon* is empty and the variable *buzzoola* is initialized
4. The variable *buzzoola* is empty and the variable *xameleon* is initialized.

In the first three cases the syntax of the resulting preroll variable is correct, but in the fourth case the syntax is wrong because the value of the preroll variable ends with an ' and', which causes a parsing error. To ensure this doesn't happen, the developers simply appended a random domain at the end of the *preroll* variable.

The appended random domain was not supposed to be contacted unless the number of valid advertising domains from the list was below a specified threshold. The developers might have added this domain either because they didn't expect this situation to happen and the domain to be contacted or because they didn't expect the domain to be sinkholed and someone else to receive these requests.

We sinkholed this random domain (321[redacted].ru) at the end of January 2020. They fixed the issue a few days later, by replacing the random domain with a valid one:

```
var player = new Playerjs({
 id:"player",
 file:"https://www.youtube.com/embed/"+id_arr_1.rand(),
  …,
  preroll: "" + imhop + "" + nativimho + "" + instream + "" + instinread + "" + instream568 + ""
+ independant + "" + independant2 + "" + adspgpmd + "" + nativ + "" + instream + "" + smby + ""
+ otm1 + "" + imho + "" + nativ2 + "" + yandex + "" + otm3 + "" + advarkads2 + "" + adriver +
"" + advarkads + "" + vihub1 + "" + vihub2 + "" + vihub3 + "" + adspektr + "" + skwstat + "" +
betweendigital + "" + republer + "" + adspmt + "" + utraff + "" + otm2 + "" + kinoaction + "" +
kinoaction2 + "" + kinoaction3 + "" + videonow + "" + vhead + "" + xameleon + "" + buzzoola +
"https://[redacted]/player/html5/media/vpaid.xml"
});
```

During the few days in which the random domain was both under our control and used by the HTML5 player's code, the sinkhole received requests from half a million unique IP addresses.

This is an example of a received request:

```
GET / HTTP/1.1
Host: [redacted]
Connection: keep-alive
Origin: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/78.0.3904.108 YaBrowser/19.12.4.25 Yowser/2.5 Safari/537.36
Accept: /
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Referer: [redacted]
Accept-Encoding: gzip, deflate, br
Accept-Language: ru,en;q=0.9
```

2.3 Available algorithmically generated domains

Advertisers started to use domain generation algorithms [3] as a fallback mechanism for the hard-coded domains – probably in an attempt to bypass ad blockers.

2.3.1 Use Case 5: Advertising mediator

This use case features a multi-platform mobile advertising mediator.

An advertising mediator interposes itself between:

- Application developers willing to integrate advertising SDKs in their applications in order to monetize their applications by showing advertisements to users of their applications
- Advertisers that want to show the advertisements of their clients in popular environments.

The mediator aims to simplify the relations between application developers and advertisers by integrating multiple advertising SDKs in a single SDK for the developers to integrate in their applications.

The mediator from this example integrates 70+ advertising providers.

It has a list of hard-coded domain names as well as a domain generation algorithm. The algorithmically generated domain names are only contacted if the hard-coded domain names are not reachable (probably when the hard-coded domains are blacklisted).

The domain generation algorithm generates one domain per year, one domain per month and one domain per day. For the generated domain names, the top level domain name is '.com' and the second level domain name is the MD5 hash computed on a seed and the date. The date contains: the year for the yearly domains; the year and the month for the monthly domains; the year, the month, and the day for the daily domains.

The sinkholed generated domains receive requests from tens of thousands of devices. Note that the number of devices contacting the sinkhole is only a subset of the total number of devices using applications that use this advertising mediator because the sinkhole is only contacted by devices that didn't manage to contact the advertiser's hard-coded domain.

This is an example of a received request:

```
POST /get HTTP/1.1
Host: [redacted]
Content-Type: text/plain
Connection: keep-alive
Accept: */*
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 13_4_1 like Mac OS X) AppleWebKit/609.1.20 (KHTML,
like Gecko) Mobile/17E262 DManager/27
Accept-Language: fr-FR;q=1, ar-FR;q=0.9, en-FR;q=0.8
```

```
Content-Length: 1064
Accept-Encoding: gzip, deflate, br
```

```
{
 "session_uptime": 7452,
 "rooted": false,
 "locale": "fr-FR",
 "connection": "wifi",
 "package_version": "1.5",
 "height": 896,
 "adidg": false,
 "user_settings": {
    "alcohol": 0,
    "gender": 0,
    "age": 0,
    "relation": 0,
    "smoking": 0,
    "occupation": 0
 },
 "ram_total": 2968862720,
 "timezone": "+0100",
 "connection_fast": false,
 "install_time": 1578266858,
 "android_id": "[redacted]",
 "battery": 39,
 "advertising_tracking": "1",
 "width": 414,
 "show_array": [
    "applovin", "inmobi", "vast", "mailru", "vungle", "mobvista", "yandex", "chartboost",
"tapjoy", "unity_ads", "startapp", "mopub", "adcolony", "appodealx"
],
 "sdk_version": "2.4.10",
 "device_type": "phone",
 "ad_stats": {
    "finish": 0,
    "click": 0,
    "show": 0
 },
 "platform": "ios",
 "lt": 0,
 "session_uuid": "163203a0-a915-11ea-bf5d-7b319b1d326c",
 "consent": true,
 "android": "13.4.1",
 "manufacturer": "apple",
 "type": "video",
 "pxratio": 2,
 "user_agent": "Mozilla/5.0 (iPhone; CPU iPhone OS 13_4_1 like Mac OS X) AppleWebKit/609.1.20
(KHTML, like Gecko) Mobile/17E262 DManager/27",
 "ram_free": 45219840,
 "idfv": [redacted],
 "ios": "13.4.1",
 "app_key": "[redacted]",
 "framework": "ios_native",
 "sdk": "2.4.10",
 "crr": "604-00",
 "coppa": false,
 "segment_id": 9670,
 "package": "[redacted]",
 "app_uptime": 1879866,
 "session_id": 450,
 "ext": {},
 "local_time": 1591577948,
 "model": "iPhone11,8"
}
```

This mediator is used in approximately 2,000 different applications. In the top 10 applications sorted by the number of requests received from them, one application has almost one million installs, one application has almost half a million installs and four applications have a quarter of a million installs.

Since this is an advertising mediator, the requests contain a lot of information about both parties it connects. On the one hand, there is information about the advertisers and the advertisements. On the other hand, there is information about the applications integrating the SDK as well as the devices on which they are installed and the users using them.

The mediator reports the list of advertisers enabled for the application in the field called show_array. The information about the advertisements contains the type of advertisement and whether advertising tracking is enabled, but also usage statistics such as the number of advertisements shown, clicks and advertisements fully watched.

Information about the devices includes:

- Hardware characteristics such as device manufacturer, type (phone or tablet) and model; height, width and pixel ratio of the display; the total and the available RAM, percentage of battery.
- Software characteristics such as platform (*Android* or *iOS*), operating system version, SDK version, whether the device is rooted or not.

Information about the applications using the SDK contains the package name and version, but also the application's install time and uptime.

Information about the users is provided through the 'user_settings' field which details the gender, age, relation, occupation and even information about whether the user is smoking or drinking alcohol. However, very few of the requests have values different from zero for these fields. Other user-related information provided in the requests are the user's language, time zone, and local time.

Given this amount of information, a potential attacker can both perform reconnaissance and gain an infection vector simply by sinkholing a mediator and analysing the received requests.

2.4 Comparison

For simplicity, we summarize the information about the use cases in Table 1.

Use case	**Applications / Websites**	**Unique IP addresses per day**	**Platform**	**Fixed**
Use case 1 (hard-coded domain)	~40	~500 000	Android	No
Use case 2 (hard-coded domain)	~10 000 websites, ~20 000 applications	>10 000 000	Android, iOS	No
Use case 3 (wrong domains)	~20 000 websites, ~1 000 applications	~2 500 000	Mobile, browser	Yes
Use case 4 (wrong domain)	Unknown	~500 000	HTML5 player	Yes
Use case 5 (DGA)	~2 000 applications	Unknown, at least ~60 000	Mobile	No

Table 1: Use cases summary.

3. POSSIBLE ATTACKS

One possible attack would be to pretend to be another legitimate mediator and let advertisers use your 'product', but use the sinkholed network to actually show the ads: no need to develop a mediator SDK and convince application developers to use it. It can even go as far as selling advertising services on the underground market.

The most dangerous attack is for an attacker to impersonate the mediator and provide its own specially crafted advertisements. The following list enumerates some possible content that can be served by an attacker in order to take advantage of the situation:

- Exploits can be sent in order to infect the device
- Products belonging to third-parties can be advertised for a fee
- Personal products can be advertised
- Scareware / ransomware content can be sent and a payment / ransom demanded
- A message (for example, a political one) can be spread

- Fake advertisements can be sent
- Phishing content can be sent.

In the case of verbose advertising SDKs which send a lot of information about the device, the application and the user from which the SDK is executing, an attacker can gather the data and sell it on underground markets. Even more, this case allows the attacker to personalize the sent fake advertisement to target a specific user's hardware and software.

For example, in the mediator use case presented earlier, the requests contain information about which versions of which applications are embedding the mediator SDK. This allows an attacker to develop, test and serve an exploit targeting that particular application version. The same goal – of targeting specific vulnerabilities – can be achieved by looking at the values of other fields provided by the presented mediator: device type, manufacturer, model, platform, operating system version, SDK version, whether the device is rooted or not, and so on.

Also, most of the requests contain the User-Agent header field, which means an attacker receiving these requests knows the User-Agent, and thus the vulnerabilities of the contacting browser. Armed with this information, the attacker can serve exploits targeting these specific vulnerabilities instead of blindly sending any exploits and probably outing himself in the process.

4. PROOF OF CONCEPT

It is not unheard of for advertisements to be used as an infection vector. However, this paper is not about the usual malvertising [2]. In usual malvertising, attackers submit malicious advertisements to advertising networks in order to infect the devices on which the advertisement is shown.

In the usual scenario, the attackers pay for their malvertisements. In this case they can infect devices through advertisements much more cheaply.

Furthermore, in usual malvertising, the attackers might not know if the devices to which their malvertisement is sent actually have the vulnerabilities that their malvertisements are exploiting. In this case, they might first learn the exploits to which each device is vulnerable and only after this send particular exploits to particular devices. Therefore this paper presents a variation of malvertising in which the attackers sinkhole the advertising infrastructure domains instead of going through the usual channels of submitting the malvertising through the advertising networks.

Let's see how such an attack would work for a mediator that uses hard-coded domains, but also domains generated by a domain generation algorithm as a fallback mechanism.

Figure 1 illustrates the case in which the mediator's hard-coded domain is accessible. In this case, the mediator SDK from the application installed on the device can contact the mediator's hard-coded domain (step 1) and receive the location of one valid advertiser out of the list of advertisers integrated by the mediator (step 2). Next, the mediator SDK contacts the received advertiser (step 3) and requests advertisements to display in the application (step 4). Even if there are fake mediators which own domains generated by the real mediator's DGA and which expect requests, they will not be contacted in this case.

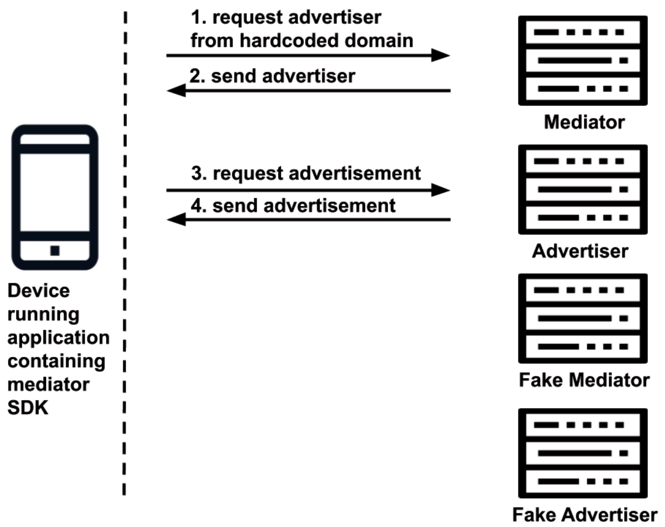

Figure 1: Communication flow when mediator's hard-coded domain is accessible.

Figure 2 illustrates the case in which the mediator's hard-coded domain is blocked and the mediator's generated domain belongs to the mediator. In this case, the mediator SDK from the application installed on the device tries to contact the mediator's hard-coded domain, but fails to do so (step 1). Upon failure, it tries to contact the mediator's generated domain (step 2). Since the generated domain belongs to the mediator, it replies with the location of one valid advertiser out of the list of advertisers integrated by the mediator (step 3). Next, the mediator SDK contacts the received advertiser (step 4) and requests advertisements to display in the application (step 5).

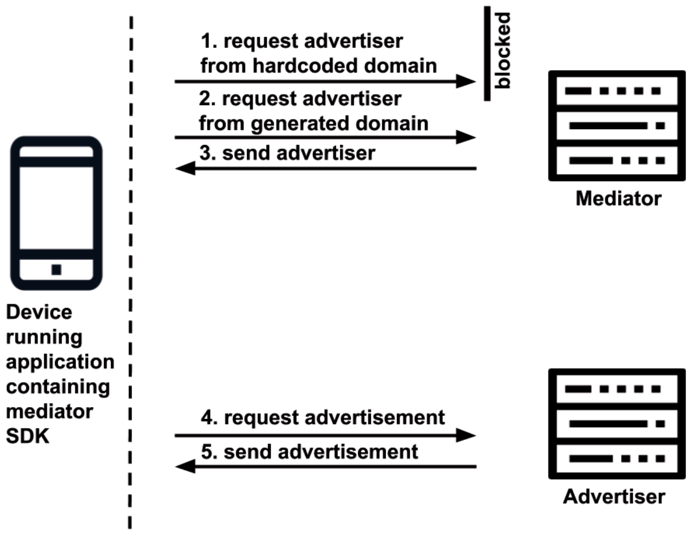
1. request advertiser from hardcoded domain
blocked
2. request advertiser from generated domain
3. send advertiser
Mediator
Device running application containing mediator SDK
4. request advertisement
5. send advertisement
Advertiser

Figure 2: Communication flow when mediator's hard-coded domain is blocked and mediator's generated domain belongs to the mediator.

Figure 3 illustrates the case in which the mediator's hard-coded domain is blocked and the mediator's generated domain belongs to a third, possibly malicious party. In this case, the mediator SDK from the application installed on the device tries to contact the mediator's hard-coded domain, but fails to do so (step 1). Upon failure, it tries to contact the mediator's generated domain (step 2). Since the generated domain belongs to a third party, the third party can reply with the location of a fake advertiser. When the mediator SDK contacts the received fake advertiser, it can receive fake, possibly malicious advertisements to show in the application. Even when the hard-coded domain belongs to the mediator and is accessible, an attacker can perform a distributed denial of service on the hard-coded domain in order to make sure that the devices contact the generated domain.

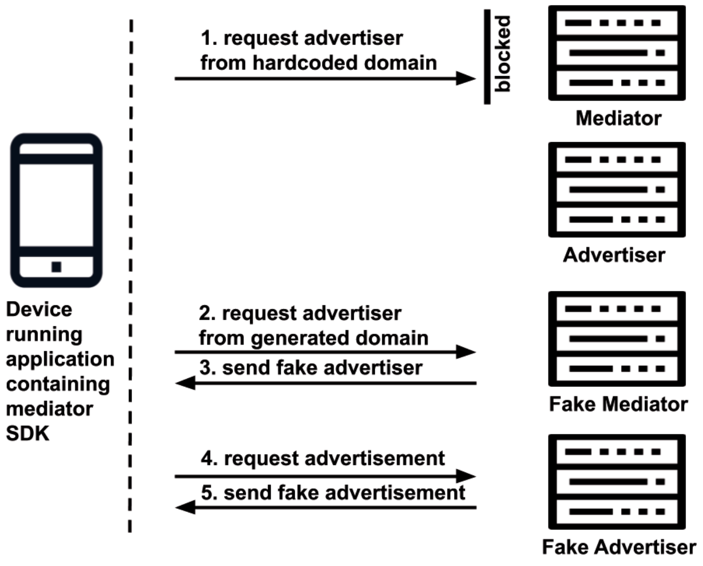
1. request advertiser from hardcoded domain
blocked
Mediator
Advertiser
Device running application containing mediator SDK
2. request advertiser from generated domain
3. send fake advertiser
Fake Mediator
4. request advertisement
5. send fake advertisement
Fake Advertiser

Figure 3: Communication flow when mediator's hard-coded domain is blocked and the mediator's generated domain belongs to a third, possibly malicious party.

In order to show the feasibility of the attack, we implemented a proof of concept for the mediator presented in the fifth use case.

We have already shown that owning a domain belonging to advertising infrastructure is easily achievable. The purpose of this proof of concept is to show that it is possible to serve specially crafted advertisements in this situation. Of course, for our proof of concept, we didn't reply to the devices contacting the sinkhole, but instead used a controlled environment. We installed one of the many applications using the SDK of this particular mediator in an emulator. For the emulator only, we redirected the generated domain that we sinkholed to localhost. On localhost, we listened for requests using a script impersonating the mediator. Since the mediator sends the location of the advertiser, we replied with a local location – thus instructing the SDK to request advertisements from a script impersonating the advertiser. From the advertiser, we served an image / video of our choosing that was shown in the application running in the emulator. The flow is illustrated in Figure 4.

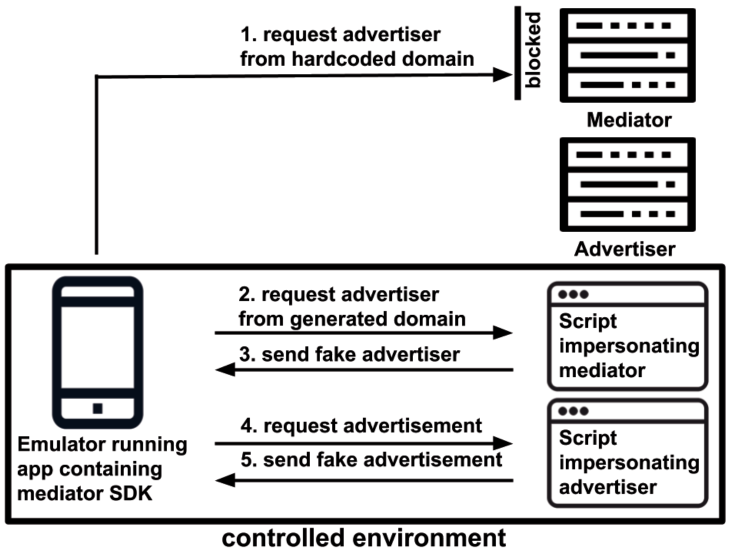
1. request advertiser from hardcoded domain
blocked
Mediator
Advertiser
2. request advertiser from generated domain
3. send fake advertiser
Script impersonating mediator
4. request advertisement
5. send fake advertisement
Script impersonating advertiser
Emulator running app containing mediator SDK
controlled environment

Figure 4: Communication flow for the proof of concept.

This attack is similar when an expired hard-coded domain is sinkholed. It is even simpler when the sinkholed domain belongs to the advertiser instead of the mediator because the attacker only needs to impersonate the advertiser, rather than both the advertiser and the mediator.

CONCLUSIONS

We believe that pointing out problems leads to improvements. Therefore, even though we don't encourage users to stop using ad blockers, we want to emphasize a situation in which using an ad blocker is more dangerous than not using it. Although the idea seems counterintuitive, such a situation is uncovered by the fifth use case presented in this paper. For this situation to happen, two conditions have to be met:

- The first condition is that the advertiser needs to use domains generated by a domain generation algorithm as a fallback mechanism for the hard-coded domains without buying all the domains the DGA generates. This means the available generated domains can be acquired and used by third parties. Since advertising providers seem to have started using DGAs in order to bypass ad blockers, such a situation is not that far fetched and we have already encountered one.
- The second condition requires only the hard-coded domains to be blocked by the user's ad blocker. Note that it is easier to blacklist hard-coded domains than to reverse engineer and implement a DGA in order to use it to regularly generate and blacklist new domains during the time interval in which they are active. Therefore, this condition is believable, too.

These two easily achievable conditions can lead to the situation described as follows. Although the hard-coded domain belongs to the advertiser and is serving advertisements, the user's device can't reach it because it is blocked by an ad blocker. The user's device contacts the generated domain belonging to a third party only because the hard-coded domain is blocked by the user's ad blocker.

Let's see why in this particular situation, it is far more dangerous to use an ad blocker than not. If the user is not using an ad blocker, he is exposed to advertisements coming from the hard-coded domains belonging to the advertiser. If the user is using an ad blocker, he can be exposed to advertisements coming from the generated domains belonging to the advertisers, as well as other things, such as exploits and malware, coming from generated domains that belong to third-party, probably malicious, entities.

Although we managed to uncover this interesting situation, it is not the main conclusion of the paper. The main conclusion is that attackers can sinkhole advertising infrastructure in order to create an infection vector targeting an advertiser's userbase. This is highly practical given the following:

- The prevalence of advertising in everyday life
- The ease with which advertising infrastructure can be sinkholed
- The fact that an attacker owning an advertising infrastructure domain can use it to serve his own specially crafted advertisements.

This is worrisome and should be taken seriously.

REFERENCES

[1] Gagliardi, P.; Lita, C. A Troubling Security Trend: Developers Letting Their Domain Infrastructure Expire. SecurityScorecard. January 2020. https://securityscorecard.com/blog/trend-developers-letting-domain-infrastructure-expire.

[2] ESET Research. Buhtrap backdoor and Buran ransomware distributed via major advertising platform. We Live Security. April 2019. https://www.welivesecurity.com/2019/04/30/buhtrap-backdoor-ransomware-advertising-platform/.

[3] Domain generation algorithm. Wikipedia. https://en.wikipedia.org/wiki/Domain_generation_algorithm.