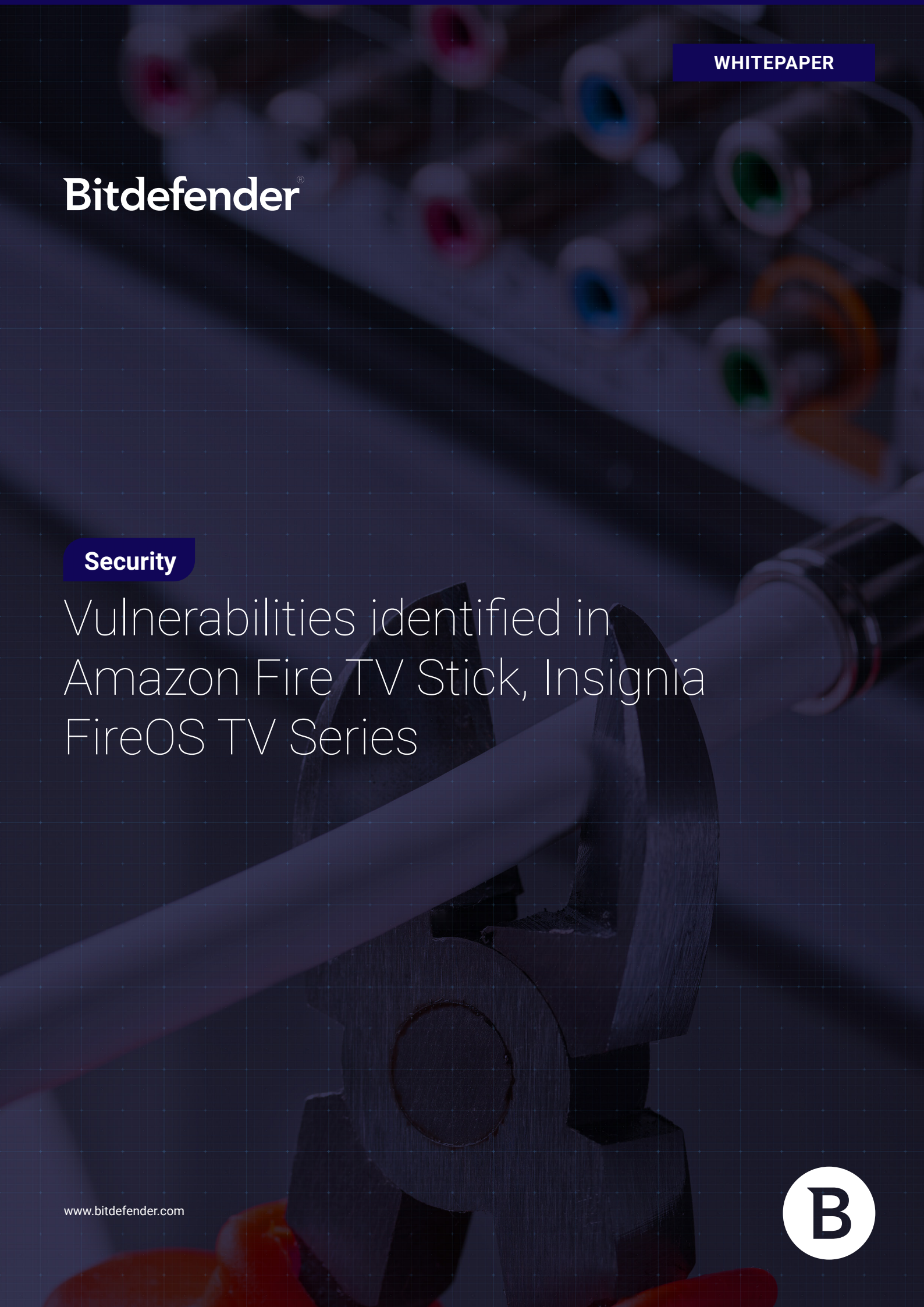
WHITEPAPER

Bitdefender

Security

Vulnerabilities identified in Amazon Fire TV Stick, Insignia FireOS TV Series



www.bitdefender.com

Contents

Vulnerabilities at a glance

- Unauthorized authentication through local network PIN brute forcing. This vulnerability was caused by improper implementation of the Password Authenticated Key Exchange by Juggling (or J-PAKE) protocol that could have resulted in attackers gaining control of the device. (CVE-2023-1385)
- A vulnerability in the setMediaSource function on the amzn.thin.pl service allowed for arbitrary Javascript code to be executed. It could be used to load arbitrary HTTP URLs in the webview. (CVE-2023-1384)
- A vulnerability in the exchangeDeviceServices function on the amzn.dmgr service allowed an attacker to register services that are only locally accessible. (CVE-2023-1383)

Disclosure timeline

Note: the vulnerabilities presented in this report have been responsibly disclosed to the vendor through their Bug Bounty program. Amazon has released fixes for these issues on Fire TV devices and the Fire TV remote app, and the company has no evidence that this issue has been used against customers. Bitdefender has been working closely with the Amazon Fire TV team through all stages of vulnerability disclosure. We would like to extend our thanks for the prompt response time, communication, transparency and escalation.

- Dec 16, 2022 - Bitdefender researchers submit the findings to the bug bounty program
- Dec 19, 2022: Bugbounty program submits the report for vendor to review
- Dec 20, 2022: Amazon team ackndowldges the findings and starts internal investigation
- Apr 12, 2023: Amazon delivers the fix to the public
- Apr 13, 2023: A bounty is awarded for the findings
- May 2, 2023: This report is published as part of coordinated disclosure

Introduction

Amazon Fire TV devices have gained massive popularity among TV users as they provide a convenient interface to transform any device, whether smart or dumb, into a fully connected device able to play local or cloud-delivered content. The functionality that allows the device to receive media content from within the local area network, and (then) play it on the TV is provided by a service named Amazon Fling Service. For applications that do not require a custom media player, the built-in media receiver is used.

This media receiver implements several functions, such as setting the URL of the media, controlling playback, or getting media information. Those functions are called through a local HTTP API, which is exposed by the WhisperPlay service. The WhisperPlay service is based on Thrift and routes messages received from clients to the correct processor - in our case, the media receiver.

To interact with the WhisperPlay API a HTTP POST request must be sent to the **/whisperlink** path. The body must contain the name of the called function as well as the required arguments, all encoded with the Thrift binary protocol.

The headers must also specify the target service ID, which in this case is **amzn.thin.pl**. This service listens on a port that is randomly selected at start-up.

Example of request headers:

```
POST /whisperlink HTTP/1.1
x-amzn-svc-uuid: amzn.thin.pl
x-amzn-dev-uuid: 2
x-amzn-svc-version: 3
x-amzn-dev-name: test
x-amzn-channel: 4
x-amzn-connection-id: 5
x-amzn-dev-type: 6
Host: 10.0.1.1
```

This service ID is specified in the documentation, but it can also be found in the **whisperplay.xml** file, which is used to declare a service for WhisperPlay.

The whisperplay.xml file from com.amazon.awvflingreceiver APK:

```
<?xml version="1.0" encoding="utf-8"?>
<whisperplay>
    <services>
        <service>
            <sid>amzn.thin.pl</sid>
            <accessLevels>
                <accessLevel>ALL</accessLevel>
            </accessLevels>
            <startService>com.amazon.awvflingreceiver.StyledMediaPlayerService</
startService>
        </service>
    </services>
</whisperplay>
```

Other WhisperPlay services can be found by searching for the **whisperplay.xml** file in all the APKs available on the device. Other services are implemented directly in the **com.amazon.whisperlink.core.android** and **com.amazon.whisperplay.contracts** APKs.

Vulnerabilities

1. **CVE-2023-1385** - Improper J-PAKE implementation allowed offline PIN brute forcing in the **com.amazon.storm.lightning.services** package, leading to unauthorized authentication

Fire TV devices can be controlled using the Amazon Fire TV smartphone app, but the app must be paired to the device first. There are two pairing methods: through the local network, when the smartphone is on the same network as the device, or remotely over the Internet. The pairing process takes place as follows:

- a randomly generated PIN is displayed on the TV
- the user must input the PIN into the app
- the app will then send the PIN to the device, that will verify it and authorize the new client.

When pairing the TV remotely, the process is handled by the **amzn.lightning** WhisperPlay service, implemented in the package **com.amazon.storm.lightning.services**, and the messages are relayed through a secure Websocket. First, the startPairing function is called, which will display the PIN on the screen. Then, to verify that the smartphone app has the same PIN, the J-PAKE algorithm is used. This protocol allows two parties to establish private and authenticated communication solely based on their shared (low entropy) password, in this case a four-digit PIN.

The security of the algorithm is based on the secrecy of two variables, X1 and X2, that should be randomly chosen by each participant. Instead of choosing them randomly, the application initializes them to one.

Functions **generateX1** and **generateX2** in **com.amazon.storm.lightning.services/org.bouncycastle.dexter.minjpake.crypto.agreement.jpake.JPAKEUtil** are called by the **JPAKEParticipant** class when pairing:

```
public static BigInteger generateX1(BigInteger q, SecureRandom random) {
    BigInteger bigInteger = ZERO;
    q.subtract(ONE);
    return ONE;
}
public static BigInteger generateX2(BigInteger q, SecureRandom random) {
    BigInteger bigInteger = ONE;
    q.subtract(ONE);
    return ONE;
}
```

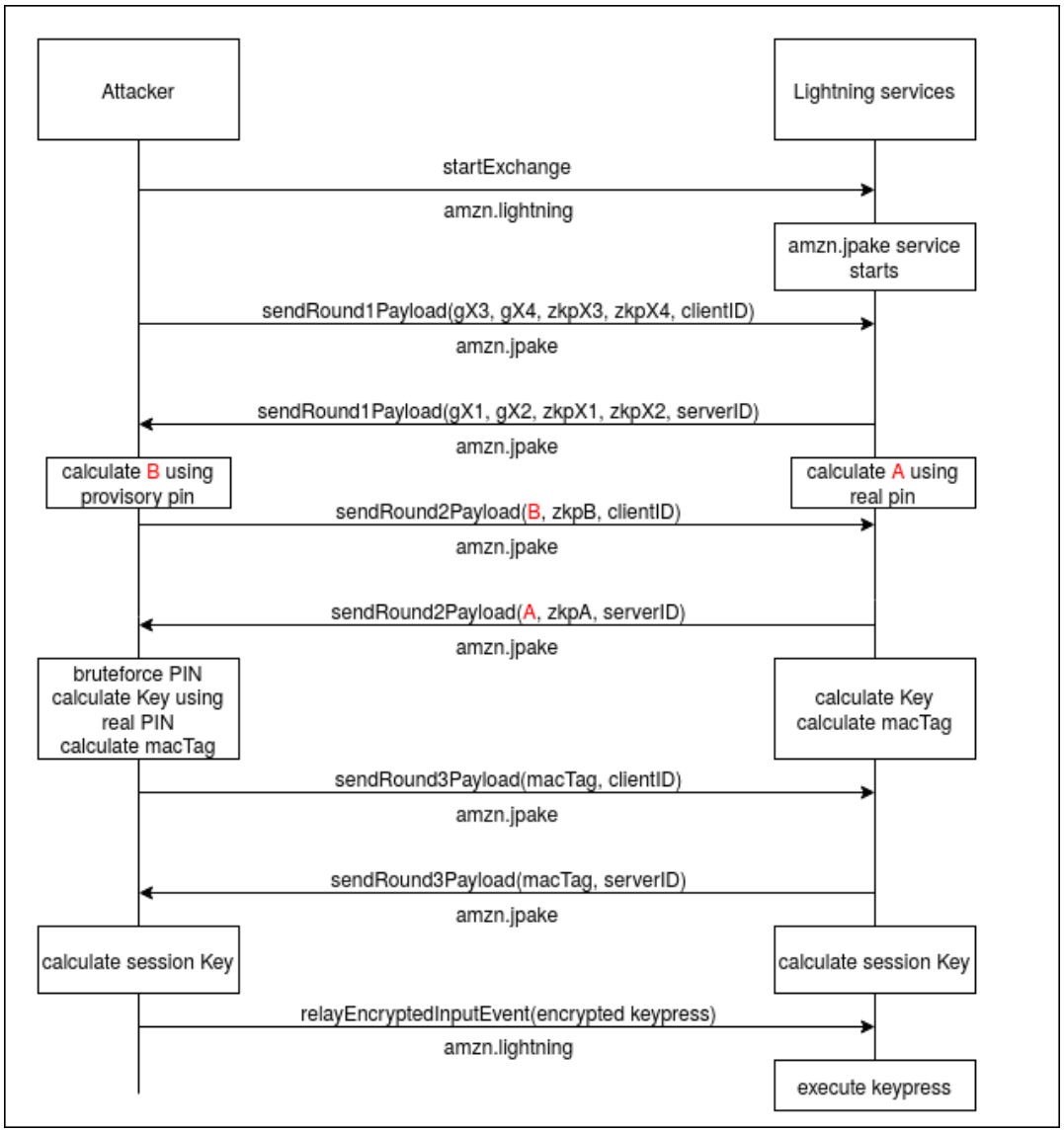
The JPAKE algorithm uses three rounds to establish a common key without disclosing information about the PIN. Of particular interest is the second round in which both parties generate and exchange a value (A) using their X2 and PIN variables:

$$A = \left(\left(g^{x1} * g^{x3} * g^{x4} \right)^{(x2 * pin)} \right) mod\ p$$

where p and g are prime numbers (the same for both parties), and x are the variables randomly chosen by the parties. Because X2 is one on the server side, the equation becomes

$$A = ((g^{x1} * g^{x3} * g^{x4})^{pin}) \bmod p$$

“g^x1”, “g^x2”, “g^x3” and “g^x4” are exchanged in round 1, and A is received in round 2. This leaves the PIN number as the only unknown and it can be brute-forced offline in at most 9000 tries. The JPAKE exchange:



Calculating the keying material for round 3:

In round 2, the client had to send an “A” value calculated with the wrong PIN number. The server will use it to calculate the final key. In a normal scenario the client would use the “A” value received from the server to calculate the key and the two keys should be identical, but because the server uses a tainted variable, this would not work. The client can instead calculate the key using the same variables as the server:

$K = (A * ((g^{x4})^{(x2*(q-pin))})^{x2}) \bmod p$, which becomes

$K = (A * ((g^{x4})^{(q-pin)})) \bmod p$

All variables are known, and the round 3 payload can be calculated using K.

Even though the local network pairing normally uses another method, the **amzn.lightning** service is also accessible locally. An attacker that is present in the local network could exploit this vulnerability and gain access to the amzn.lightning service. One possible attack outcome would be enabling developer mode on the device and starting adb, giving the attacker access to a limited shell, and the possibility to sideload applications.

2. **CVE-2023-1384** - The **setMediaSource** function on the **amzn.thin.pl** service previously did not sanitize the **"source"** parameter, allowing for arbitrary javascript code to be run

The **amzn.thin.pl** service allows a local client to play videos in a webview. The **setMediaSource** function uses the **loadUrl** function to inject javascript code into the webview without sanitizing the source. A payload in the "source" variable of the form `'http://url/\',"{\\"a\\":\\"a\\"}",true,false);(injected code);//` will execute the injected code. The URL must be valid, but it does not have to point to a video file.

The responsible code in **com.amazon.awvflingreceiver/.StyledMediaPlayer.setMediaSource:**

```
        this.mInjectedJavascriptMediaSource = "javascript:flingAdapter.
setMediaSource('" + source + "', " + JSONObject.quote(metadataJson) + ", " +
autoPlay + ", " + playInBg + ")";

        this.mSource = source;

        this.mMetaDataJson = metadataJson;

        if (this.mWebView != null) {

            this.mHandler.post(new Runnable() {

                @Override // java.lang.Runnable

                public void run() {

                    if (StyledMediaPlayer.this.mWebView != null) {

                        StyledMediaPlayer.this.mWebView.
loadUrl(StyledMediaPlayer.this.mInjectedJavascriptMediaSource);

                    }

                }

            });

        }
```

The webview cannot load intent URLs and does not possess cookies for the **amazon.com** domain. It can read files that the com.amazon.awvflingreceiver has access to and send them to another server as well as load arbitrary HTTP URLs.

3. **CVE-2023-1383** - The **exchangeDeviceServices** function on the **amzn.dmgr** service previously allowed an attacker to register services that are only locally accessible

By supplying a valid **deviceServices** structure and a valid explorerId (e.g. **tcomm**) to the **exchangeDeviceService** function, an attacker could have obtained access to the **amzn.reg**, a**mzn.auth.in**, **amzn.endpoin**t, a**mzn.act.reg**, **amzn.wpen**.**brok** and **amzn.state** services. The target device UUID is needed to construct the **deviceServices** structure and could be obtained by calling the **getFullDeviceInfo** function on **amzn.dmgr**.

About Bitdefender

Bitdefender is a cybersecurity leader delivering best-in-class threat prevention, detection, and response solutions worldwide. Guardian over millions of consumer, business, and government environments, Bitdefender is one of the industry's most trusted experts for eliminating threats, protecting privacy and data, and enabling cyber resilience. With deep investments in research and development, Bitdefender Labs discovers over 400 new threats each minute and validates around 40 billion daily threat queries. The company has pioneered breakthrough innovations in antimalware, IoT security, behavioral analytics, and artificial intelligence, and its technology is licensed by more than 150 of the world's most recognized technology brands. Launched in 2001, Bitdefender has customers in 170+ countries with offices around the world.

For more information, visit https://www.bitdefender.com.

Bitdefender

Founded 2001, Romania

Number of employees 1800+

Headquarters

Enterprise HQ – Santa Clara, CA, United States

Technology HQ – Bucharest, Romania

WORLDWIDE OFFICES

USA & Canada: Ft. Lauderdale, FL | Santa Clara, CA | San Antonio, TX | Toronto, CA
Europe: Copenhagen, DENMARK | Paris, FRANCE | München, GERMANY | Milan, ITALY | Bucharest, Iasi, Cluj, Timisoara, ROMANIA | Barcelona, SPAIN | Dubai, UAE | London, UK | Hague, NETHERLANDS

Australia: Sydney, Melbourne

UNDER THE SIGN OF THE WOLF

A trade of brilliance, data security is an industry where only the clearest view, sharpest mind and deepest insight can win — a game with zero margin of error. Our job is to win every single time, one thousand times out of one thousand, and one million times out of one million.

And we do. We outsmart the industry not only by having the clearest view, the sharpest mind and the deepest insight, but by staying one step ahead of everybody else, be they black hats or fellow security experts. The brilliance of our collective mind is like a **luminous Dragon-Wolf** on your side, powered by engineered intuition, created to guard against all dangers hidden in the arcane intricacies of the digital realm.

This brilliance is our superpower and we put it at the core of all our game-changing products and solutions.

Bitdefender-PR-Whitepaper-AMZFr-creat6696-en_EN
02/22/23April 19, 2023 6:54 pm04/21/23