CS 253: Web Security

Local HTTP server security
The most dangerous code you run every day

// Anyone can connect to the server at http://<your-ip>:4000
server.listen(4000)

// Only your device can connect to the server
server.listen(4000, '127.0.0.1')
Zoom Zero Day: 4+ Million Webcams & maybe an RCE? Just get them to visit your website!

A vulnerability in the Mac Zoom Client allows any malicious website to enable your camera without your permission. The flaw potentially exposes up to 750,000 companies around the world that use Zoom to conduct day-to-day business.

CVE-Numbers

- DOS Vulnerability — Fixed in Client version 4.4.2 — CVE-2019–13449
- Information Disclosure (Webcam) — Unpatched — CVE-2019–13450

UPDATE — July 9th (am)

As far as I can tell this vulnerability also impacts Ringcentral. Ringcentral for their web conference system is a white labeled Zoom system.
"This vulnerability allows any website to forcibly join a user to a Zoom call, with their video camera activated, without the user's permission"

"On top of this, this vulnerability allowed any webpage to DOS (Denial of Service) a Mac by repeatedly joining a user to an invalid call"

"Additionally, if you’ve ever installed the Zoom client and then uninstalled it, you still have a localhost web server on your machine that will happily re-install the Zoom client for you, without requiring any user interaction on your behalf besides visiting a webpage. This re-install ‘feature’ continues to work to this day"
"Let me start off by saying having an installed app that is running a web server on my local machine with a totally undocumented API feels incredibly sketchy to me"

"Secondly, the fact that any website that I visit can interact with this web server running on my machine is a huge red flag for me as a Security Researcher"

"Having every Zoom user have a web server that accepts HTTP GET requests that trigger code outside of the browser sandbox is painting a huge target on the back of Zoom"
Demo: How does a site communicate with a local HTTP server?
Demo: How does a site communicate with a local HTTP server?

- With the following local HTTP server:

```javascript
const COMMAND = 'open /System/Applications/Dictionary.app'

app.get('/', (req, res) => {
  exec(COMMAND, err => {
    res.set('Access-Control-Allow-Origin', '*')
    if (err) res.status(500).send(err)
    else res.status(200).send('Success')
  })
})
```

- Any site can send a GET request to `http://localhost:4000` to launch the Dictionary application.
Demo: How many servers are running on your computer?
Demo: How many servers are running on your computer?

```bash
$ lsof -i -P | grep -i "listen"
```

<table>
<thead>
<tr>
<th>Service</th>
<th>PID</th>
<th>Username</th>
<th>Type</th>
<th>IP Address</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>rapportd</td>
<td>408</td>
<td>feross</td>
<td>4u</td>
<td>IPv4 0x97025599d3aa176b</td>
<td>TCP *:57054 (LISTEN)</td>
</tr>
<tr>
<td>rapportd</td>
<td>408</td>
<td>feross</td>
<td>5u</td>
<td>IPv6 0x97025599ed02e613</td>
<td>TCP *:57054 (LISTEN)</td>
</tr>
<tr>
<td>CommCente</td>
<td>421</td>
<td>feross</td>
<td>26u</td>
<td>IPv6 0x97025599ed02b513</td>
<td>TCP [2907:fa90:5c0:906e:a1a0:f0b3:9732:fa7a]:5060 (LISTEN)</td>
</tr>
<tr>
<td>Spotify</td>
<td>27013</td>
<td>feross</td>
<td>62u</td>
<td>IPv4 0x9702559a0aa233db</td>
<td>TCP *:57343 (LISTEN)</td>
</tr>
<tr>
<td>Spotify</td>
<td>27013</td>
<td>feross</td>
<td>64u</td>
<td>IPv4 0x97025599ee8ae3db</td>
<td>TCP *:57621 (LISTEN)</td>
</tr>
</tbody>
</table>
TrendMicro local HTTP server
Remote Code Execution (RCE)

- Local HTTP server was vulnerable to RCE from any site
- See Google Project Zero issue: https://bugs.chromium.org/p/project-zero/issues/detail?id=693&redir=1
Problems with Zoom's local server

- Any site, not just zoom.us, can send a GET request to open the app and join the user to the given conference
- Conference host can decide to automatically enable video for participants
- The local server remains installed after the user uninstalls Zoom and it has the ability to re-install Zoom
- Vulnerable to UI denial-of-service
Schedule a Meeting

Topic
Jonathan Leitschuh's Zoom Meeting

Date
7/6/2019  ~  11:00 PM ~  to  ~  7/6/2019  ~  11:30 PM ~

Time Zone
(GMT-04:00) Eastern Time (US and Canada)

Recurring meeting

Video
Host  On  Off
Participants  On  Off

Audio
Telephone  Computer Audio  Telephone and Computer Audio
Dial in from United States  Edit

Options
Require meeting password
Advanced Options

Calendar
iCal  Google Calendar  Outlook  Other Calendars

Schedule
Zoom UI denial-of-service

// It's actually better if this number isn't a valid zoom conference number
const confNum = '694138052'

setInterval(() => {
  new Image().src = 'http://localhost:19421/launch?action=join&confno=' + confNum + '&' + Date.now()
}, 1)
Zoom defends use of local web server on Macs after security report

Local web server will also reportedly reinstall Zoom if a user removes the application and joins a meeting.
Zoom reverses course to kill off Mac local web server

Less than a day after backing its approach to get around Safari restrictions on Mac, Zoom’s local web server is no more.
Cleaning up the mess

- Zoom issued an updated app which uninstalled the local HTTP server and added a new UI prompt to confirm that you want to join a meeting
- User who did not open the app for a while would be vulnerable until they installed the update
- Users who previously uninstalled Zoom would not get the update, so they'd be stuck with the vulnerable local server
Remote Code Execution (RCE)

- Around 1 week after the local server issue came to light, another research team discovered a RCE vulnerability.
- The complete exploit allowed a zero-interaction RCE just by visiting a malicious site – yikes!
Apple is silently removing Zoom’s web server software from Macs

For users who haven’t seen all the drama

By Dieter Bohn | @backlon | Jul 10, 2019, 7:12pm EDT
Apple takes steps

- macOS has a silent update mechanism for disabling specific executable files (Malware Removal Tool)
- No OS update required – checks for new banned executables in the background, regularly
- Useful for disabling fast-spreading malware or vulnerable software affecting lots of users
Zoom doesn't understand CORS
User joins a zoom call (vulnerable)
Local Server
localhost:19421

Client

GET /j/123 HTTP/1.1

Server
zoom.us
GET /j/123 HTTP/1.1

HTTP/1.1 200 OK
<!doctype html>

Page is loaded

GET /launch?action=join&confno=123 HTTP/1.1

Initiated by zoom.us

Local Server
localhost:19421

Launch Zoom app

Server
zoom.us
GET /j/123 HTTP/1.1
HTTP/1.1 200 OK
<!doctype html>
Zoom doesn't understand how CORS works?

- The `http://localhost:19421/launch?action=join&confno=###` endpoint returns information about whether the request succeeded, but since it's triggered from `https://zoom.us` the same origin policy doesn't allow reading the response.

- So, they returned an image with different widths/heights to "leak" information to the site that triggered the request.

- They could have just used `Access-Control-Allow-Origin` to specify particular sites which would be allowed to read the response.
User joins a zoom call (with CORS endpoint) (vulnerable)
GET /j/123 HTTP/1.1

HTTP/1.1 200 OK
<!doctype html>

Page is loaded
GET /j/123 HTTP/1.1

HTTP/1.1 200 OK
<!doctype html>

Page is loaded

GET /launch?action=join&confno=123 HTTP/1.1

Initiated by zoom.us

Launch Zoom app
Attacker joins user into a zoom call
GET / HTTP/1.1

HTTP/1.1 200 OK
<!doctype html>

GET /launch?action=join&confno=123 HTTP/1.1

Initiated by attacker.com

Page is loaded
Attacker joins user into a zoom call (with CORS endpoint)
Let's fix the issue

- **Best solution:** remove the local HTTP server and just register a `zoom://` protocol handler

- However, let's assume we need to keep the local HTTP server (probably a bad idea)
  - How can we secure it?

- Ideas:
  - Require user interaction before joining, don't allow host to automatically enable video
  - Only allow `zoom.us` to communicate with the local server
User joins a zoom call (local server inspects Origin header)
GET /j/123 HTTP/1.1
Host: zoom.us

HTTP/1.1 200 OK
<!doctype html>

GET /launch?action=join&confno=123 HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us

HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}

Is origin zoom.us?
Yes

Launch Zoom app

Page is loaded

Initiated by zoom.us

CORS response
GET /j/123 HTTP/1.1
Host: zoom.us

HTTP/1.1 200 OK
<!doctype html>

Page is loaded

GET /launch?action=join&confno=123 HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us

Initiated by zoom.us

Is origin zoom.us?
Yes

HTTP/1.1 200 OK
Access-Control-Allow-Origin: zoom.us
{"result": "success"}

Origin is allowed

CORS response

Launch Zoom app

Local Server
localhost:19421

Server
zoom.us
Attacker joins user into a zoom call (local server inspects Origin header)
GET /launch?action=join&confno=123 HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com

Page is loaded

HTTP/1.1 200 OK
<!doctype html>
GET /launch?action=join&confno=123 HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com

HTTP/1.1 200 OK
<!doctype html>

Page is loaded

Initiated by attacker.com

Is origin zoom.us?
GET /launch?action=join&confno=123 HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com

HTTP/1.1 200 OK
<!doctype html>

Local Server
localhost:19421

Server
attacker.com

Page is loaded

Initiated by attacker.com

Is origin zoom.us?
No

Close socket
Problems with inspecting Origin header

- **Problem:** Browser doesn't always add Origin header
  - for "simple" requests (e.g. `<img>` or `<iframe>` tags)
  - for same origin requests (e.g. `fetch()` to same origin)
  - Very old browsers

- **Solution:** block requests where Origin header is omitted

- **Solution:** change the endpoint to require a "preflighted" request so that Origin header is always sent (e.g. change the HTTP method to PUT)
"Simple" HTTP requests

- An HTTP/1.1 **GET, HEAD** or a **POST** is the request method
- In the case of a POST, the **Content-Type** of the request body is one of **application/x-www-form-urlencoded**, **multipart/form-data**, or **text/plain**
- No custom HTTP headers are set (or, only CORS-safelisted headers are set)
"Preflighted" HTTP requests

- Before a "preflighted" requests can be sent to the target server, the browser must check that it is safe to send.
- So it first sends an HTTP request with the `OPTIONS` method to the same URL.
What happens if the browser does not preflight "non-simple" requests
GET / HTTP/1.1

Client

Server attacker.com

Server victim.com
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> Attack code

Attacker page loads
GET / HTTP/1.1

HTTP/1.1 200 OK
<!doctype html> Attack code

DELETE /item/42 HTTP/1.1
Cookie: sessionId=123
GET / HTTP/1.1

HTTP/1.1 200 OK
<!doctype html> Attack code

DELETE /item/42 HTTP/1.1
Cookie: sessionId=123

Delete item 42
GET / HTTP/1.1
HTTP/1.1 200 OK
<!doctype html> Attack code

DELETE /item/42 HTTP/1.1
Cookie: sessionId=123
HTTP/1.1 200 OK
<!doctype html> Deleted!
Access-Control-Allow-Origin: victim.com
Introducing the OPTIONS request

- Browser sends OPTIONS request first to ask the server if the request we want to send is okay
- If server doesn't support OPTIONS (either because it is old or because it doesn't want to support preflighted requests) then, preflighted requests are denied
- Let's see how it can protect our local HTTP server
User joins a zoom call (local server requires "preflighted" request)
Browser

PUT /launch ... HTTP/1.1
Host: localhost:19421

Initiated by zoom.us

GET /j/123 HTTP/1.1
Host: zoom.us

HTTP/1.1 200 OK
<!doctype html>
GET /j/123 HTTP/1.1
Host: zoom.us

HTTP/1.1 200 OK
<!doctype html>

PUT /launch?... HTTP/1.1
Host: localhost:19421

Browser

Initiated by zoom.us

Local Server
localhost:19421

Server
zoom.us

Client
GET /j/123 HTTP/1.1
Host: zoom.us

HTTP/1.1 200 OK
<!doctype html>

PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us

OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://zoom.us
Access-Control-Request-Method: PUT

Preflight request

Is origin zoom.us?
Yes
HTTP/1.1 204 No Content
Allow: PUT

Preflight response

PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by zoom.us

Req not same origin
Attacker joins user into a zoom call (local server requires "preflighted" request)
Local Server
localhost:19421

Initiated by attacker.com

GET / HTTP/1.1
Host: attacker.com

HTTP/1.1 200 OK
<!doctype html>

PUT /launch?... HTTP/1.1
Host: localhost:19421
GET / HTTP/1.1
Host: attacker.com

HTTP/1.1 200 OK
<!doctype html>
GET / HTTP/1.1
Host: attacker.com

HTTP/1.1 200 OK
<!doctype html>

PUT /launch?... HTTP/1.1
Host: localhost:19421

Initiated by attacker.com

Req not same origin
GET / HTTP/1.1
Host: attacker.com
<!doctype html>

HTTP/1.1 200 OK

PUT /launch?... HTTP/1.1
Host: localhost:19421
Initiated by attacker.com

OPTIONS /launch?... HTTP/1.1
Host: localhost:19421
Origin: https://attacker.com
Access-Control-Request-Method: PUT

Preflight request

Browser

Is origin zoom.us?
No

HTTP/1.1 403 Access Forbidden

Preflight response

Req denied
Who can still launch the app from the local server?

- Preflight requests seems to allow the local server to distinguish requests from **zoom.us** and those from random sites.

- However, other native apps running on the same device can still fool the local server.
  
  - The browser enforces that sites can't tamper with the **Origin** header, but a native app (e.g. a Node.js or Python script) can make a request and set the **Origin** header to **https://zoom.us**.
One more thing...

- Every site on the web can send requests to our local HTTP server!
  😞
  - Works against the server that required "preflighted" requests as well as the server which just checked the \texttt{Origin} header

- Next time... we'll discuss DNS rebinding attacks ✨
Credits:

https://medium.com/bugbountywriteup/zoom-zero-day-4-million-webcams-maybe-an-rce-just-get-them-to-visit-your-website-ac75c83f4ef5