Outsmarting the Smart City

Daniel Crowley, Mauro Paredes, Jennifer Savage
August 2018

Introduction

Cities change with the needs and desires of their inhabitants. In an era when individuals are purchasing myriad Internet of Things devices for their Smart Homes, many people are taken by the promise of a Smart City that is connected in similar ways. Cities have responded by deploying Smart City technology where convenient, or to solve specific needs.

Smart City Technology is everywhere once you know how to look for it. The devices are in boxes bolted to street posts, on top of buildings, and on city vehicles. Sometimes they are highly visible, such as internet connected parking meters that allow individuals to pay their parking with their cell phone. Sometimes they are less obvious, such as hidden sensors designed to detect flooding or radiation. Someday, people will look at cities without all these bells and whistles as being stuck in another time.

This research examines the security of a cross-section of the devices currently in use today. Traditional reconnaissance and application testing techniques were used to reveal how deeply flawed many of these devices are.

Example Uses of Smart City Technologies

Disaster Management - Sensors exist to detect a wide array of natural disasters, which help citizens get to safety faster and notify first responders of potential problem areas. Some of the sensors available detect radiation, floods, earthquakes, wildfires, hurricanes, tsunamis, and tornadoes.

Surveillance - Cameras embedded in street lights are less obvious than the traditional models and allow citizens to be followed visually as required, such as when a gunshot sound is detected. Information about phones and other bluetooth devices present in an area is gathered in strategic places.

Resource Management - Smart meters on homes, such as water and electric, help reduce the amount of manual meter reading required and allow city utilities to cope with an expanding

population without adding addition meter reader personnel. Smart water meters can detect the source of a leak based on where water use has spiked.

Traffic Management - Smart city devices exist for traffic monitoring and control, including signal controllers for emergency vehicles, route planning, traffic optimization, and emergency warnings. Smart parking meters allow citizens to pay their toll using their cell phone, while also providing useful information about what meters are currently in use. Smart toll collection eliminates the need for toll booths. Vehicle to Infrastructure Communication hubs allows connected vehicles to be notified of traffic jams and accidents.

City Reconnaissance

Reconnaissance on cities can be performed with simple passive techniques and open source intelligence. Manufacturers of smart city devices do case studies on the implementation of their devices within cities as a way of advertising their capabilities. Smart cities are often proud of the technology they have deployed, so they advertise it. A search for news reports on a particular city's smart city technologies will turn up a great deal of information. Many cities also make their contracts public, which will quite often provide detailed information about what has been deployed.

The IANA (Internet Assigned Numbers Authority) ranges assigned to cities are searchable online, which can make it much simpler to find IP addresses to use when searching websites such as Shodan or Censys. Both Shodan and Censys act as search engines for networks and devices, making it simple to identify what is being run and where.

Physical reconnaissance involves walking or driving around and looking for these devices. They are often simply mounted on poles on the side of the street, or placed on top of publicly accessible buildings such as parking garages.

Source code for devices may be made available online through websites such as Github and OSADP, facilitating independent code audits. For devices that do not have open source software and firmware, updates are often downloadable and may be reversed to learn more about the inner workings of the device.

Device Security

Echelon i.LON 100 / i.LON SmartServer and Echelon i.LON 600

Echelon describes the i.LON 100, now called the i.LON SmartServer, as “a versatile controller, router, and smart energy manager that connects control devices to IP-based applications such as building automation, enterprise energy management, demand response programs, and high-value remote asset management programs.”

Attack Scenario

The Web and FTP passwords for the i.LON device family are default values out of the box, and don’t need to be changed, but when they are changed they must be changed individually. Changing the Web password doesn’t change the FTP password, and vice versa. If both passwords have been changed from their default values, however, the API for the i.LON 100 / i.LON SmartServer does not require authentication with the default configuration. This API, in older versions, allows the username and password for the FTP server to be retrieved. In newer versions, the FTP credentials cannot be retrieved from the device, but they can be changed through the API.

Once authenticated to the FTP server, an attacker can replace the files on the device to change the way it functions, which could be used to gain persistent access to the device, add a new Web user to access the functionality provided by the Web server, and sabotage the device. The i.LON 600 has a secure default authentication configuration, but the i.LON family suffers from an authentication bypass bug. When a Web request is made, the path requested (e.g. /dir/file.txt for http://example.com/dir/file.txt) is compared character by character against the paths in the configuration file to see if the i.LON should ask for a username and password. However, if a single slash in a requested path is replaced with two or more slashes, this will prevent the i.LON from matching against the configured paths. Once the path is requested at the operating system level, the extra slashes will be removed and the file will be served normally.

There’s an additional challenge to exploiting the i.LON 600, though. Even with an authentication bypass, the i.LON 600 runs in a restricted mode by default which prevents most changes from being made. In order to put it in the open mode that allows all settings to be changed, a person must physically be present at the device to push a button while it boots. While this prevents an attacker from altering the FTP credentials to be able to replace the binaries on the device, it does not prevent them from changing the IP address of the device, locking legitimate users out of the device until they reset the device while pushing a button on the device itself with a paperclip.

Vulnerabilities

Default Configuration Allows Authentication Bypass
Affects: i.LON 100 / SmartServer

Issue: The default `WebParams.dat` file for the i.LON 100/i.LON SmartServer restricts access only to `/forms/Echelon/*`, but sensitive configuration items can be changed through use of the SOAP API at `/WSDL/iLON100.WSDL`, including the FTP username and password.
Exploitation: An attacker can invoke API calls directly without the need for authentication and change the FTP credentials for the device, then replace the system binaries with malicious versions.
Recommended Fix: Change the default `WebParams.dat` file to enforce authentication on `/WSDL/*` as well as `/forms/Echelon/*`

Authentication Bypass
Affects: i.LON 100 / SmartServer, i.LON 600
Issue: Authentication is controlled by configuration directives in the WebParams.dat file. When specifying that a particular set of files or directories should be inaccessible without authentication, the path is placed in the configuration file as a string with optional wildcard character (*) to match zero or more characters. No path canonicalization is applied before comparing paths.
Exploitation: By issuing Web requests with superfluous slashes in the URI (e.g. `/forms////Echelon/SetupSecurity.htm`) the path will not match the one configured to require authentication and will be accessible without any username or password:
Recommended Fix: Canonicalize received paths before matching them to ones specified in the configuration.

Default Credentials
Affects: i.LON 100 / SmartServer, i.LON 600
Issue: i.LON 100 and i.LON 600 devices come with default credentials of `ilon/ilon`. No password change is required on initial setup.
Exploitation: Attackers can log in with publicly known credentials if credentials are not changed upon installation.
Recommended Fix: Require users to change the default credentials upon installation.

Plaintext Passwords
Affects: i.LON 100 / SmartServer, i.LON 600
Issue: The passwords in the `WebParams.dat` configuration file are in plaintext.
Exploitation: An attacker who has gained access to the i.LON device can retrieve the `WebParams.dat` file and obtain the passwords for all users.
Recommended Fix: Store the passwords in hashed format.

Unencrypted Communications
Affects: i.LON 100 / SmartServer, i.LON 600
Issue: Multiple services do not use encryption to protect communications. Web servers are run unencrypted by default, and configuration and firmware updates are delivered via FTP.
Exploitation: An attacker with the ability to observe a user authenticating to the device can capture credentials being sent in plaintext over the network.

Recommended Fix: Move to encrypted protocols to protect sensitive data in transit.

Battelle Vehicle to Infrastructure Hub

The Batelle V2I Hub software allows connected vehicles to interface with transportation infrastructure. It facilitates translation of data from multiple protocols and sources used in transportation, allowing them to process messages that are key to connected vehicle applications such as red light violation warnings. Through the use of Traveler Information Messages (TIM), the system can also develop and send infrastructure information about advisory speeds, attributes of physical elements, such as bridge heights, and other data that can be used by applications, such as curve speed warnings and over-height warnings.

Attack Scenarios

In V2I Hub v2.5.1, which has been deprecated along with the rest of v2.x in response to our discoveries, there is a hard-coded administrator account which cannot be removed without modifying the code. While it's unlikely this will be changed in any real-world deployment, if it is, there is still an API. This API requires a key, and if the default has been changed, it's possible to request the API key directly through the web server, without authentication. In one of the API endpoints, there is a SQL injection flaw, which can be used to retrieve credentials for the rest of the application.

In V2I Hub v3.0, there is a SQL injection flaw in the login process, which allows attackers to retrieve credentials from the database directly, without authentication.

Vulnerabilities

V2I Hub v2.5.1

Sensitive Functionality Available Without Authentication
Issue: A PHP script available from the web server shuts down the system immediately by running the halt command as root.
Exploitation: An attacker can simply visit `http://V2I_HUB/UI/powerdown.php` to shut down the system on which V2I Hub is running.
Recommended Fix: Enforce authentication on `powerdown.php`

Hard-Coded Administrative Account
Issue: An administrative account is hard-coded into V2I Hub that cannot be disabled without altering the source code.

Exploitation: An attacker can log in as an admin on any installation of V2I Hub with the username `_battelle` and password `B@ttelle`.
Recommended Fix: Remove the hard-coded account.

Default API Key
Issue: There is no requirement to change the default API key used by V2I Hub upon installation.
Exploitation: Attackers can use all available API functions on V2I Hub installations where the default API key has not been changed.
Recommended Fix: Generate a random API key on first use.

API Key File Web Accessible
Issue: The API key file is accessible from the Web server.
Exploitation: Attackers can visit `http://V2I_HUB/api/apikey.txt` to obtain the current API key for a V2I Hub installation, even if it has been changed.
Recommended Fix: Move the API key outside the web root.

API Auth Bypass
Issue: The API key is checked directly against a user-supplied value in PHP's `$_GET` global variable array, using PHP's `strcmp()` function followed by a comparison with `0` using the equality operator `==`. The `strcmp()` function exhibits some unexpected behavior when it is used to compare a string and an array: It returns `NULL`. The `==` operator returns true when comparing `0` and `NULL`. Moreover, when the name of a GET variable is followed by open and close square brackets, such as `key[]`, PHP interprets it as an array.
Exploitation: An attacker can execute API functions without the correct API key by adding `[]` to the end of `key` in the URL when accessing API functions. An example can be seen at `http://V2I_HUB/api/PluginStatusActions.php?action=list&key[]=any_value_works_here&jtStartIndex=0&jtPageSize=20&jtSorting=name+ASC`
Recommended Fix: Use PHP's strict equality operator, `===`, instead of `strcmp()`.

Reflected XSS
Issue: User-supplied input is placed in the context of the HTTP response without sanitization, and can be used to manipulate the way the browser interprets the resulting response and insert arbitrary scripting into the page. This can be used to launch a number of attacks, including session hijacking and phishing attacks.
Exploitation: An attacker who can trick a user who is logged into V2I Hub into visiting a website under their control can execute arbitrary Javascript in the browser of that user, taking any actions available to the user and accessing all information available to the user.
Recommended Fix: When accepting input from users, input should be subject to strict validation on the server side using either whitelisting or a "known good input" filter. Whitelisting involves creating a list of acceptable characters that can be part of an input. For instance, US zip codes should only ever include numbers and dashes, and should be of the format 12345 or 12345-6789. A "known good input" filter restricts input to a set of possible good inputs. For instance, if valid parameter values can only ever be "blue" or "red", "mauve" should never be

accepted. This strategy is more secure than whitelisting, but in most cases is impractical due to the number of possible valid inputs.

SQL Injection
Issue: When including user input in the context of code, it is important to make a clear distinction that the user input is to be interpreted as data. Without a clear distinction, specially crafted input may cause part of the data to be treated as code, allowing users to execute arbitrary code. Several locations were found where user data was concatenated into SQL queries.
Exploitation: Freely available tools such as SQLmap can be used to discover and exploit SQL injection flaws such as the ones found in V2I Hub. The following is an example of one way that data may be extracted from the database using SQL injection flaws that can be observed in a web browser (this displays the database version, name, and logged in database username in the page):

```
http://V2I_HUB/status/pluginStatus.php?id="-1"or+exists(select+count(*),concat(((select+concat(0x53714c69,mid((concat(concat_ws(0x04,version(),database(),user()),0x01030307)),1,64)))),floor(rand(0)*2))from+information_schema.tables+group+by+2)&name=ivpcore.MessageProfiler--
```

Recommended Fix: Parameterize all SQL queries. Do not concatenate any user input into SQL queries, even in seemingly harmless clauses such as `ORDER BY` and `SORT BY` clauses.

V2I Hub v3.0

SQL Injection
Issue: When including user input in the context of code, it is important to make a clear distinction that the user input is to be interpreted as data. Without a clear distinction, specially crafted input may cause part of the data to be treated as code, allowing users to execute arbitrary code. The log in functionality of V2I Hub v3.0 was found to be vulnerable to SQL injection in the username.
Exploitation: An attacker can manipulate the login query to return unrelated data from the database one bit at a time. This includes the username and password of all users in the database.
Recommended Fix: Parameterize all SQL queries. Do not concatenate any user input into SQL queries, even in seemingly harmless clauses such as `ORDER BY` and `SORT BY` clauses.

Libelium Meshlium

Libelium describes the Meshlium as, a “IoT Gateway to connect any sensor
to any Cloud Platform”. Advertised uses for the platform include cell phone detection, radiation monitoring, flood prevention, and agricultural automation.

Attack Scenario

Exploiting the Meshlium involves sending a shell injection payload to one of a series of endpoints that do not check authentication and which feed user input into shell commands in an unsanitized way. This allows any shell command to be executed as the “www-data” user, which controls the web server. This user is also given passwordless sudo access, meaning that it can execute any command as root, a user with complete control over the entire system. Attackers can now take a variety of actions, including attacking other systems accessible by the Meshlium, inserting fake sensor data to disrupt systems that rely upon correct sensor data, or selectively removing sensor data to prevent important events from being detected.

Vulnerabilities

Shell Injection
Issue: Shell injection is a type of security flaw that is present when improperly sanitized user input is used to construct a shell command which is then executed. Several scripts were found to be in place on Meshlium devices that execute shell commands that include unsanitized user input without authentication, specifically:

```
/ManagerSystem/plugins/f_instaler/updates/server.php
/ManagerSystem/plugins/e_system/f0_backup/php/upload.php
/ManagerSystem/presets/MeshliumRF3GAp/defaults/etc/.fr-td0JMI/Manager
System/upload.php
/ManagerSystem/upload.php
```

Exploitation: Attackers can use various shell metacharacters to change the nature of shell commands being executed. For instance, backticks can be used to execute arbitrary commands as shown in the proof of concept exploit below:

```
<!-- start of exploit -->
<form
action="http://MESHLIUM_IP/ManagerSystem/plugins/f_instaler/updates/s
erver.php" method=”post”>
<input type="hidden" id="type" name="link" value="downloadUpdate" />
<input type="hidden" id="link" name="link" value="`touch
/tmp/vulnerable`" />
<input type="submit" value="Launch proof of concept exploit" />
</form>
<!-- end of exploit -->
```

To use the exploit above, replace `MESHLIUM_IP` with the IP address of a running Meshlium system, and save it as an HTML file. Open the file and press the button shown. This will create a file on the Meshlium at `/tmp/vulnerable`.

Recommended Fix: Avoid including user input in shell commands. If user input must be included in shell commands, use whitelisting to allow only known safe characters such as letters and numbers

Implications

Due to the severity of the vulnerabilities discovered, any threat actor capable of communicating directly with one of these vulnerable devices is capable of using them in the same way as legitimate administrators can. It's also possible to use these devices to pivot into any network the device is connected to, which may allow access to other smart city devices or sensitive systems.

Cities should be aware of the risk that the intended functionality of a device could be subverted if the device is vulnerable. Safety mechanisms such as cameras or backup sensors should be used to verify sensor readings used for critical decisions such as evacuations. Considerations also should be made for situations where a vulnerable system is reporting normal readings when it should not be, or situations where there are abnormal readings when everything is normal.

The privacy concerns raised by the use of smart city technologies are heightened with the risk of their attack by malicious threat actors. Many smart city devices collect, or at least have access to sensitive data. Given data from license plate readers, for instance, a car can be tracked throughout a city.

Conclusion

Smart city technology solves a number of problems, but it comes with the potential for security and privacy problems. When security is not well considered in the design, creation, deployment, and maintenance of smart city devices, city infrastructure carries a heightened risk of the devices having vulnerabilities that may be taken advantage of.

With a greater reliance on smart city devices, there comes a greater risk that the abilities granted by these devices may be abused. The same capabilities intended to be used by city administrators to enrich a city may be useful for other purposes in the hands of a threat actor. There is a risk that citizens will fear that their city's infrastructure is insecure.

If cities intend to continue on the current path of connecting their infrastructure and making decisions regarding important things based on these connected devices, there should be more in-depth testing of the security of the devices. The public should be assured that these devices have been tested, the test information should be made publicly available after disclosure and patching, and recommendations should be made regarding reducing the privacy invasions inherent to mass data collection by these devices. Smart City device vendors should be expected to take security into consideration throughout the development lifecycle and obtain in-depth independent security testing of their devices prior to sale. Cities should be advised to not purchase devices that have not received this testing.