SCADA and mobile: security assessment of the applications that turn your smartphone into a factory control room

20.05.2015

This document is a preliminary revision.
For the latest version, please see
http://github.com/Darkkey/AndroidHMISecurity

Authors:

Ivan 'Steph' Yushkevich
(i.yushkevich@dsec.ru)
&&
Alexander 'dark_k3y' Bolshev
(@dark_k3y, abolshev@dsec.ru)
Digital Security Research Group

Table of contents

0. Introduction

The days when mobile technologies were just a rising trend have passed, and now mobile devices are an integral part of our life. As a result, you may find them in places where they probably shouldn't be. But convenience often wins over security. Nowadays, you can monitor (or even control!) your ICS (Industrial Control System) from a brand-new Android or iOS smartphone. Just type the words 'HMI', 'SCADA', or 'PLC' into Google Play Store or ITunes App Store, and a surprisingly large bunch of results will appear. Moreover, many of these applications are developed by serious vendors, like Siemens, GE, Omron, etc., and allow accessing, monitoring, or controlling the HMI, PLC, DCS, or SCADA systems in your ICS infrastructure. Are they secure? Could an attacker do something bad if they get access to an industrial engineer's tablet? What kind of vulnerabilities can exist in these applications? What attack vectors are possible?

This paper tries to answer some of these questions. The main goal of our research was not only to find security bugs in ICS-related mobile applications, but also to try and extrapolate the risks of compromising these applications to the risks of compromising whole ICS infrastructure. This point of view differs from usual mobile application security assessment point of view: bugs, that could introduce low threat, in the ICS environment could lead to a high risk; and vice versa, bugs that usually marked as high threat, will have low or very low probability to lead to something dangerous in case of ICS environment.

1. ICS 101

Before we proceed to describing ICS mobile applications and associated threats, we need to do a quick intro into the modern ICS infrastructures. ICS is a general term which unites software and hardware complexes used in industry automation, including distributed control systems (DCS), supervisory control and data acquisition systems (SCADA), and other systems, such as PLC, HMI, MES, etc. Nowadays, every factory, plant, business center, railroad, or even your house is controlled by some kind of ICS. According to the general definitions, ICS collects data from remote stations, processes them and uses automated algorithms or operator-driven supervisory to create commands to be sent to remote devices (also called field devices). First ICS infrastructures appeared in the 1970-80. The nature of such systems leads to rare updates and slower progress of technology. However, in the last decade, modern things like XML, .Net, JSON, etc. are used in ICS more and more. And of course, mobile applications are also in game.

Modern ICS infrastructures have complex architectures that consist of commonly known elements, such as servers, PCs, network switches, software technologies (.Net, DCOM, XML, etc.) and more scary things, such as PLCs, transmitters, actuators, analog control signals, etc. Let's look at a sketch of the modern ICS infrastructure (see Fig. 1). We can see three basic levels in Fig. 1.

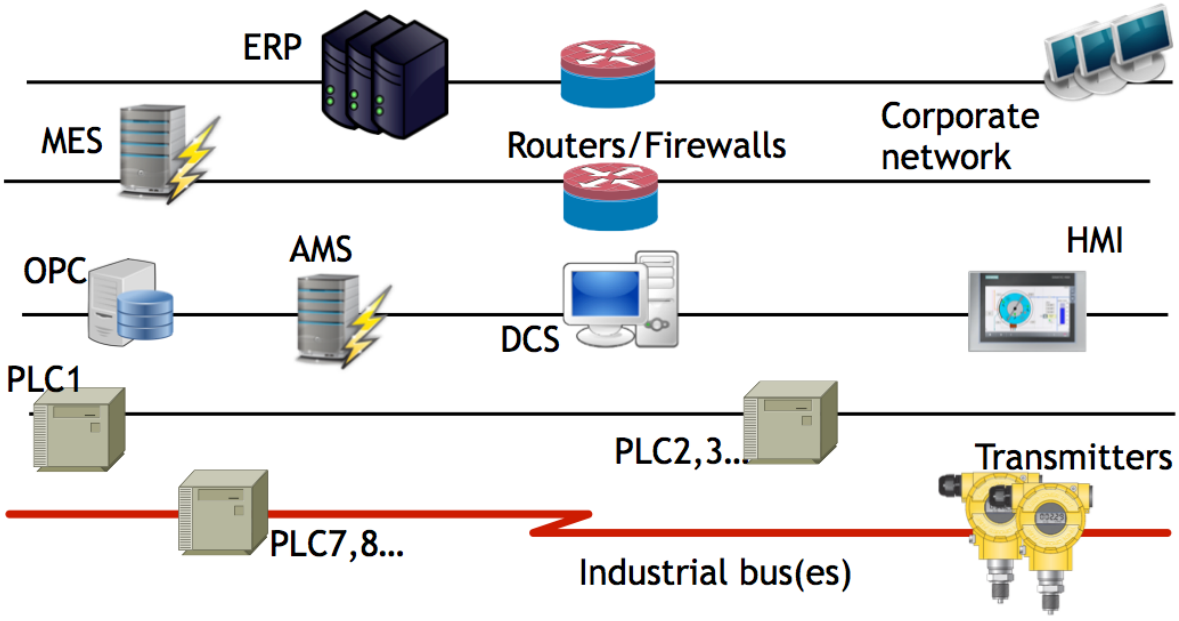

Figure 1: Modern ICS infrastructure sketch

1. **Lower level**, where field devices are located. As it was mentioned earlier, these devices are suitable for the dirty work – for example, they can monitor temperature and pressure in a reactor, control such operations as opening and closing valves, turning pumps on and off, and other things. Multiple devices can be used on this level. It may be a low-level Programmable Logic Controller (**PLC**, according to Wikipedia [1]: computer-based, solid-state device that controls industrial processes). Also, transmitters and actuators, controlled by Remote Terminal Units (**RTU**, microprocessor-controlled electronic device that interfaces objects in the physical world to a distributed control system or SCADA (supervisory control and data acquisition) system by transmitting telemetry data to a master system, and by using messages from the master supervisory system to control connected objects[2]), can be used in this level. This level is the kingdom of low-level industrial protocols, such as Modbus or Modbus TCP, HART, Wireless HART, Profibus DP or PA, Foundation Fieldbus H1, and others. Low-level ICS engineers, electricians, technicians, and other staff work with ICS on this level.

2. **Medium level**, where we have high-level PLCs, Distributed Control System (**DCS**, control system, usually of a manufacturing system, process or any kind of dynamic system, in which the controller elements are not central in location (like the brain) but are distributed throughout the system with each component sub-system controlled by one or more controllers[3]) and Supervisory Control and Data Acquisition (**SCADA**, systems operating with coded signals over communication channels so as to provide control of remote equipment (using typically one communication channel per remote station)[4]) systems, Human-Machine Interface (**HMI**) controlling stations and various servers such as Open Platform Communications (**OPC**, earlier OLE for Process Control) server. All intellectual work takes place on this level. Here, based on the data from the lower level, operators or automated systems decide what to do and send commands to field-level devices. The entire industrial automation process occurs here. Operators, process engineers, ICS engineers, PLC and software programmers work with systems on this level.

3. The **upper level** refers to the integration of business with industrial processes. This layer provides bindings to corporate networks and Enterprise Resource Planning (**ERP**) systems. Various Plant Asset Management (**PAS**) and Manufacture Execution Systems (**MES**, which provide the right information at the right time and show the manufacturing decision maker "how the current conditions on the plant floor can be optimized to improve production output."[5]) work on this level. Management and top-level engineering staff work with ICS on this level.

[1] http://en.wikipedia.org/wiki/Programmable_logic_controller.
[2] Gordon R. Clarke, Deon Reynders, Edwin Wright, Practical modern SCADA protocols: DNP3, 60870.5 and related systems Newnes, 2004, ISBN 0-7506-5799-5, pages 19-21.
[3] http://en.wikipedia.org/wiki/Distributed_Control_System.
[4] http://en.wikipedia.org/wiki/SCADA.
[5] McClellan, Michael (1997). Applying Manufacturing Execution Systems. Boca Raton, Fl: St. Lucie/APICS.

2. Types of mobile ICS applications and associated risks

As you know now, ICS infrastructures are vast and various. Thus, mobile applications for ICS are also have various purposes. Because there are (yet) no classifications of such applications, we introduce our classification, based on the relations with industrial process and location of such application inside ICS architecture. We identify three types of mobile ICS applications (see. Fig. 2):

- **Control-room apps (1)**, with the purpose of direct configuring/monitoring/supervising industrial process and/or its components. There are several roles of such applications in the ICS infrastructure:
    - **PLC configuration app.** Configuring and/or monitoring the state of ICS components, such as PLCs, HMIs connected to PLCs, RTUs, and other things. In this case, Mobile ICS application connects to the ICS component over the industrial network (low and middle level of infrastructure). The purpose of such application is to monitor the state of a component or to (re)configure the component. You could think of such an application as a standard handheld terminal, but inside your Nexus or Xperia. Example application: SIEMENS LOGO! App.
    - **SCADA client.** Supervising the industrial process as a SCADA client or VNC to SCADA interface. These applications allow an industrial engineer to use their phone or tablet to connect to the SCADA application and supervise the industrial process in case of need. Example application: ProficySCADA.
    - **Mobile HMI panel.** Create an HMI panel inside your mobile device to monitor or control several industrial components. Applications of this type allow an engineer to design (and even program!) the HMI panel interface and connect its components to the real PLCs or other devices over Modbus/TCP, OPC, and other protocols/interfaces. Example application: ScadaTouch Basic (HMI-Modbus).

  There is one important thing that unites all such applications -- connections between app and industrial component occur in the allegedly “safe” environment, somewhere near low-middle levels of ICS. Therefore, lack of cryptography, authentication, and other “usual” things could not be classified as a high risk. More dangerous things could happen if the server (ICS component) part of the application doesn’t check the correctness of input (in terms of industrial process) or, even worse, contains bugs that could lead to DoS or other consequences. Also, such applications, if they store data in some kind of Android storage, should not allow third parties to modify it. Elsewhere, unexpected industrial process behaviour could occur (e.g., imagine that a mobile HMI panel saves information about gauges that control some components to the SD card. If the attacker could somehow modify this information, for example, swap the components between two gauges, this could lead to the situation when the engineer thinks they are looking at one temperature gauge but is actually seeing the state of another part of the ICS infrastructure).

- **OPC/MES/Historian client (2).** These applications allow the engineer and process owner to read and interpret some data from middle-high level components of ICS. The data is read-only; moreover, you don't have direct access to the PLCs, HMIs or SCADA apps -- your only ability is to read some variables. Also, we included mobile clients for MES apps into this category. The typical usage of such applications is to read some data from the historian server or some aggregated data from MES system. A typical representative of this type is the OPC client/browser. They don't connect directly to the ICS component, but rather connect to OPC server. The main connection point of such a mobile application is the high levels of infrastructure. Also, the connection could be done remotely. Thus, more crypto- and auth-like mechanisms should be implemented in the app. While reading aggregated data itself couldn't directly lead to any threat, this data could be used by attacker to understand (reverse-engineer) the industrial process and craft a specialized attack. Also, when an application is used outside of the plant network, new threats appear, like compromising the engineer/manager's phone through a vulnerability in the application. Example application: OPC XML DA Explorer.

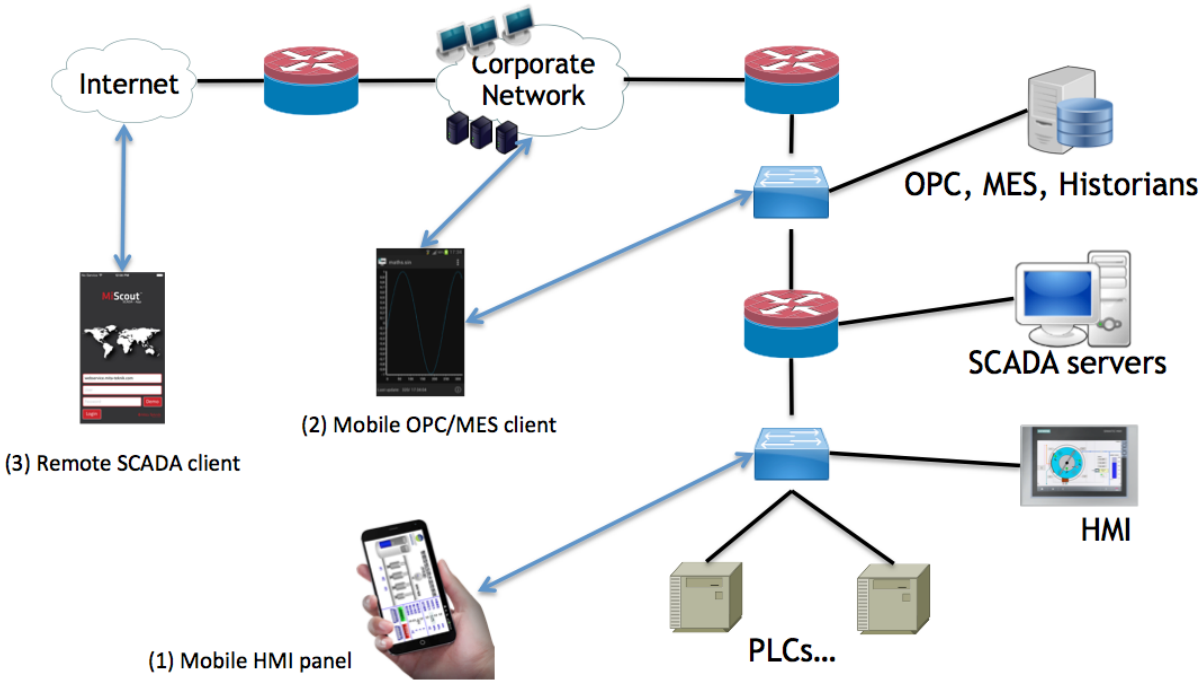

Figure 2: Typical places of mobile ICS applications

- **SCADA remote control app (3)**. Applications that allows remote (outside of the plant network) monitoring/controlling of the industrial process. While applications from the first type could also

be used for such activities, it is not recommended or even showed by the vendors. For **ALL** applications in this group, we found pictures/schemes/architecture sketches/documents from the vendor where the mobile app is shown as a remote control client ***outside*** of the plant network (high-low levels). Therefore, we think that potential users of such application will occasionally (or usually) connect to the industrial process from public networks, untrusted home networks, or using cellular networks. The Android environment of such application is also unclear and could potentially include malicious software, rooted platform or other bad things. Security and reliability requirements to applications of this type are the highest. Example application: Pro-face Remote HMI**.**

In the following table, you can see potential threats for the mobile application depending on its type.

App. type.	**Threats**
Control-room app	● server DoS or compromise ● lack of server-side data validation in terms of industrial process ● compromise of stored data that could lead to interface/feature modification (HMI apps) ● client-side DoS
MES/OPC/Historian client	● process information leak through a protocol vulnerability ● server DoS or compromise ● client-side DoS or compromise ● deceiving the operator for hiding alarms
SCADA remote control app	● compromise of process through a protocol or application vulnerability ● lack of server-side data validation in terms of industrial process ● compromise of a process through a server vulnerability ● client-side DoS or compromise

3. How and what we tested

Analysis. During our analysis of each application, we've done some common checks. Using the acquired results, we were able to evaluate the the application in security terms. Each test check provides some useful information or actual bugs which became points for further analysis (e.g. to do some advanced checks or try specific attack vectors). After completing all tests, we started more specific analysis of applications in terms of safety.

Our test checklist:

1. **Authentication**.
    - Whether some authentication exists in the application or it allows communicating with server/PLC/etc without any logins or passwords.
    - What kind of information is exchanged during the authentication process between the mobile app and server/PLC/etc.? How is the authentication process organized? What protocol is used, are there any cryptographic mechanisms?
2. **Does the application have any password protection (that somehow restricts access to its functionality)?** For example, if a user somehow loses their smartphone (or it is stolen) with an installed mobile SCADA application and it is found by an attacker, what could prevent attacker from immediately connecting to SCADA/PLC/etc.? If there is no default password protection, could it be turned on?
3. **Where and how are passwords/connection strings stored?**
    - Whether they are stored in shared_preferences or on the SD card.
    - Are there any encryption/hashing/other cryptographic primitives in case? Imagine an attacker got full access to the smartphone or access the configuration files (e.g. virus or data leak through other channels, like another application or platform bug) -- how fast could they get the critical information?
4. **What permissions are required by the application?** When you install an application, you need to give it the permission to access specific capabilities or information on your device. For example: access data on the SD card, Wi-Fi connectivity, SMS and phone services access, etc. This could be useful if the application could be compromised -- how could this fact affect other applications on the mobile device? Also, by analysing these rights, you could find out whether the application collects some extra info from your phone (when it shouldn't).
5. **Does the application contain any native code?** Unlike Java, native code applications can have much more problems such as buffer overflows, use-after-free and other dangerous vulnerabilities.
6. **Are there any web-based components in the application?** Inaccurate usage of such components could lead to web vulnerabilities. Thus, we need to do specific web checks for these applications.

7. **What is the main purpose of the application and what protocols are used -- SCADA / HMI / PLC / OPC / etc.?**
8. **Are there any cookie/tokens or something like that?** Are there any specific identifiers in the application requests? Where exactly is authentification made -- on the server or in the application itself?
9. **Does the application use SSL?** It is necessary to check the configuration of SSL. For example, does the application check the correctness of certificate or does it have certificate pinning? This check lets us know whether we are able to construct any specific attack associated with SSL (e. g. if any SSL incorrect configuration exists, could it be used for an MitM attack?)
10. **Does the application use XML?**
    - What parser is used to process XML data?
    - What configuration does it have? This information helps to find out are if any XML-related attacks can be done.
11. **What information does the app store?** Connection strings to server, HMI project files, logs and other information that may be interesting to the attacker or could contain other important data. What format is used for data storage? Is this information encrypted and/ or can it be easily accessed?
12. **What server API is available to the application?** Or, in other words, which features/capabilities are available for the application when it is communicating with the server?

When we finished all these checks, the more specific analysis of the application began. It includes understanding its logic, control flow, and other important things.

Fuzzing. In some cases, if the application protocol is binary and very hard to reverse-engineer or understand, we used mutational-based fuzzing to find flaws. In the Fig. 3, you can see the fuzzing architecture that we used. Application connects to server (PLC, SCADA, MES, OPC, etc.) through a transparent socket server, which occasionally mutates passing data. For example, it could mutate data from client to server with the probability of 5 % or 25%. So, mutation-based fuzzing checks are embedded in the normal data flow and this allows us to check random places of protocol parsing on both sides.

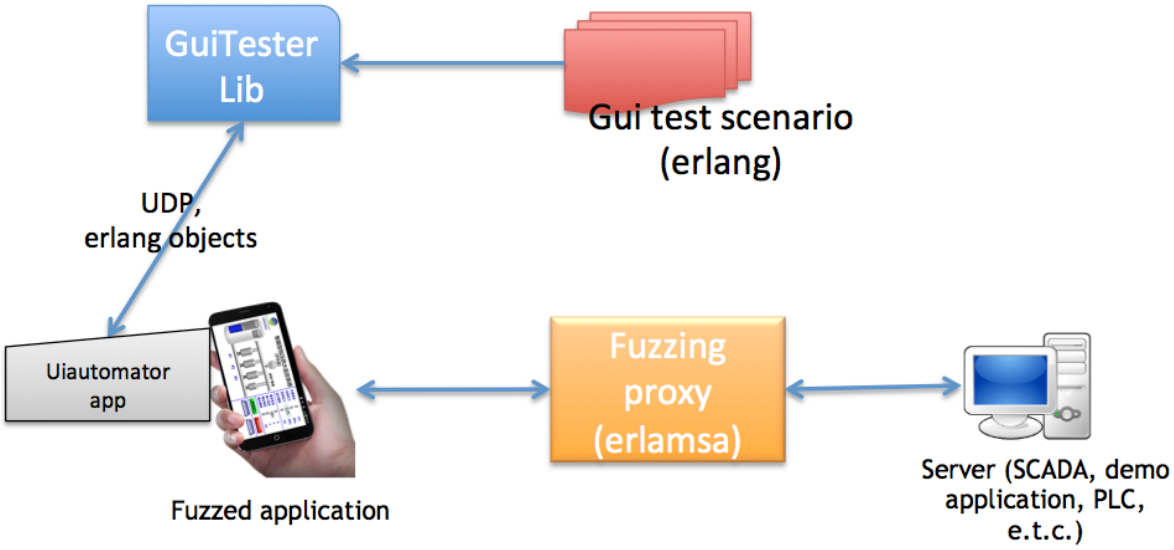

Figure 3: Fuzzing infrastructure

We're in the mobile word, and automated fuzzing of *client* application is hard. Moreover, there is another problem -- some of the fuzzed application has *native interface components*. This leads to the situation where you could not use automated UI testing tools to click application controls to force it to send/receive some data from server. To solve this problem, we've developed a special tool called AndroidNUITst (Android Native UI TeSTer). This tool consists of two components: Java application that uses adb and Android UIAutomator library to emulate screen taps and capture screenshots; the second component is erlang service that uses test scenario to trigger some application functions and detect crashes. For example, here is the test scenario for Siemens LOGO! App:

```
-module(logoscript).

-export([prepare/2, fuzz/1]).

show_status_window(S) ->
      {ok} = nuitestclient:send_cmd(S, {click, 200, 600}, 3000),
      {ok} = nuitestclient:send_cmd(S, {click, 200, 350}, 3000),
      {ok} = nuitestclient:send_cmd(S, {click, 200, 200}, 3000),
      {ok} = nuitestclient:send_cmd(S, {setcropscreenheight,
692}, 3000).

run_logoapp(S) ->
```

```
            {ok} = nuitestclient:send_cmd(S, {ping}, 3000),
            {ok} = nuitestclient:send_cmd(S, {home}, 3000),
            {ok} = nuitestclient:send_cmd(S, {run,
"com.siemens.snc.ilogo/qAndroidDevice.QAndroidDevice",
"android.intent.action.MAIN"}, 3000),
            timer:sleep(20000),
            show_status_window(S).


      prepare(IP, Port) ->
            S = nuitestclient:connect(IP, Port),
            run_logoapp(S),
            timer:sleep(5000),
            {ok} = nuitestclient:send_cmd(S, {putscreenoncache,
"working"}, 5000),
            S.

      fuzz(S) ->
            fuzz(S, true, 0).

      fuzz(_, _, 30) ->
            "Too many fails. Crashed?";
      fuzz(S, true, _FailCnt) ->
            {ok, Res} = nuitestclient:send_cmd(S,
{comparescreenoncache, "working"}, 5000),
            io:format("Got ~p from android.~n", [Res]),
            fuzz(S, Res, 0);
      fuzz(S, false, FailCnt) ->
            io:format("Relauncing...~n"),
            {ok} = nuitestclient:send_cmd(S, {click, 358, 463}, 3000),
            timer:sleep(1000),
            show_status_window(S),
            timer:sleep(1000),
            {ok, Res} = nuitestclient:send_cmd(S,
{comparescreenoncache, "working"}, 5000),
            io:format("Got ~p from android.~n", [Res]),
            fuzz(S, Res, FailCnt + 1).
```

Here, we send commands to the server like *home* (press phone Home button), *click* (tap a specific screen area), *run* (run an application), and others. To control the application flow, and detect crashes (or disconnects), there is a screenshots cache. Remember that we are dealing (in most cases)

with native application components, and we cannot use usual UIAutomator functions. Instead of this, at first iteration of the fuzzer we do a clear run (without any protocol mutations), capture app screenshots and put them in the cache. On the next runs, we periodically compare screens in the cache against the current screen -- and if we detect any differences, a crash or disconnect has probably occured and we need to log it and restart the application.

Applications list. In the following table, you will find all mobile applications (classified by type) that we have tested:

Application	Vendor	Server	Type
Afcon Pulse	Afcon	Pulse Server	SCADA remote control app, MES app
Autobase SCADA	Autobase	Autobase SCADA systems	SCADA remote control app
CyBroDroid	Cybrotech	Cybrotech PLCs	SCADA remote control app
Ellat SCADA	Ellat	ELLAT WEB-SCADA	SCADA remote control app
HMI MASTER	-	Any PLC with Modbus	Control room app
HMI OBA7	-	Siemens S7 PLCs	Control room app
Siemens LOGO! App	Siemens	LOGO PLC	Control room app
MiScout	MiScout	MiScout Web server	SCADA remote control app
OPC XML DA client	-	Any XML OPC DA server	OPC client
OPC XML DA Explorer	-	Any XML OPC DA server	OPC client
PLC-5 Mobile HMI Express	-	Allen-Bradley PLC-5/40e	Control room app
Pro-face Remote HMI	Proface	Proface PLCs	SCADA remote control app
ProficySCADA	GE	iFIX, CIMPLICITY	SCADA remote control app

Prosys OPC UA Client Lite	-	Any OPC UA server	OPC client
S7 Android	-	Siemens S7 PLCs	Control room app
Movicon Progea Web CLient	Movicon	Movicon Progea SCADA	SCADA remote control app
sScadaTouch	-	Any PLC with Modbus	Control room app
Wagoid	-	WAGO (and compatible) PLC with Modbus	Control room app
Watch*IT		WatchIT OPC Server	OPC client
WHS Live Light	-	WHS	Historian client

4. Test results

All found bugs can be placed into several categories. We will describe some of these categories and then move on to the examples and overall statistics.

Authentication & Authorization (A&A) bugs

More than 40 % of all reviewed applications have problems with A&A or access control problems.

Client-side authentication. For example, in some mobile applications you could get information from server and do several actions without registration on the server. Such mobile apps send a static formed key and login/password pair at the time of A&A. The only purpose of this key is device identification (or it could be used by the server for logging). If authentication succeeds, server will return nothing except the constant -- “yes, authentication success”. There is no correct logout procedure and there is no real session handling by server. If the server will close connection, the application will have no reaction to it. If you exit the mobile application, it will continue working in the background, accepting server notifications. If you know the device ID and this user has passed A&A, you could send commands from another device and they will be accepted and processed by server. Thus, the A&A mechanism just protects the mobile application, but not access to the server.

Absence of any A&A mechanisms. About 30 % of tested applications allow reading and writing data (or executing commands) in the SCADA system without ANY authentication. As it was mentioned above, this could be a low-risk flaw for control-room applications, but it should not happen at all for mobile applications with other purposes, especially for remote SCADA clients. However, it was the most common problem. You could get HMI panel designs, infrastructure circuits, system and process states, alarm notifications etc -- all without authentication. This information could be very useful for the attacker, because they could get information about the industrial process or partly reverse-engineer it. And all they need is just knowledge about the target system’s IP address! Moreover, in some applications you could write values or even execute commands! Fig. 4 shows the percentage of applications that allow reading/writing without authentication (applications that work with PLCs are excluded from this list).

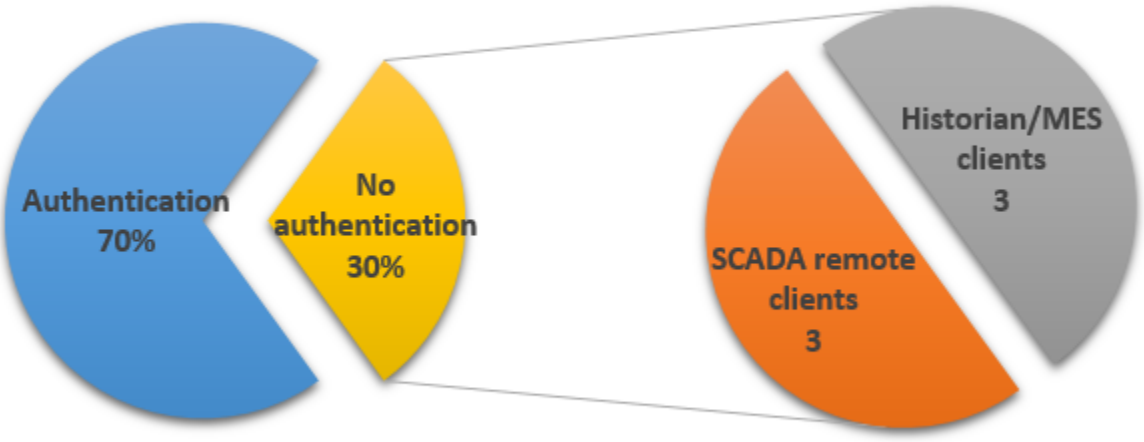

Figure 4: Authentication usage in mobile ICS applications

Plain-text authentication. Several apps use no encryption or hashing during authentication process, so credentials are sent in clear text. This could allow attacker (if they have access to the channel, e. g. open Wi-Fi, or MitM in Wi-Fi, or a GSM channel) to get a full-rights connection to the system with no additional efforts.

Weak hashes. Weak hashing is another discovered problem. One application used unsalted MD5 hashing, which isn't very strong for the modern bruteforce engines and computer speed. Another "good" approach was the self-invented hashing algorithm, which is based on several symbols, operations, and typecasting. These leads to only 2^{25} hash variations (33000000). The chance of collision is very high, and and modern systems can break such a hash in a matter of seconds. As in the previous case, using weak hashes simplifies the life of the attacker. If they could capture some traffic with the authentication information, restoring the original data could take seconds (custom hashes) or several days (MD5).

Other authentication bugs. Also, we have found a bunch of other authentication bugs, for example, one remote SCADA client authentication process requires a username (password isn't sent during the session).

Protocol-related weaknesses

No crypto. All data between client and server is transmitted in clear text. As it was shown above (see **Plain-text authentication)**, this could lead to bad consequences in case of an untrusted channel. If

the attacker can monitor the channel between, they can gain sensitive process and system data. If the attacker can spoof the replies from client or server (Man-in-the-Middle attack), they can:

- Send fake data about industrial process to the mobile application;
- Send fake commands to the server;
- Extend attack vector, e. g. by sending bad/unexpected data to client/server. If there are any Denial of Service vulnerabilities, this could lead to a crash.

Incorrect usage of SSL/TLS. One of the main goals of SSL usage is to prevent MitM attacks. Every SSL connection begins from the handshake procedure, when the server sends its certificate to the client. SSL/TLS connection security vastly depends on the correctness of the certificate check procedure. The most common source of errors -- incorrect configuration of this functionality, e. g.:

- turning off certificate checking;
- absence of hostname against certificate check;
- no SSL pinning.

All of this could compromise the handshake procedure and allow an attacker to MitM the connection.

Weak cryptographic keys storage. As we've said earlier, critical data transmission and storage should be encrypted. Several applications have problems with storage of the encryption keys. If the attacker gets access to the keys, they could easily decipher traffic or stored data. 20 % of applications with secured data storage have precompiled keys inside the native or Java binaries. Here are some key samples: "ZXCVB...", "42@DSOTM#...", etc. This is equivalent to having no data storage encryption.

SCADA-related bugs

Insufficient checks for process-correct values. Several applications have no server-side check for the industrial process correctness of the variable value. E. g. you could set the temperature to 32000 or to -1000. All checks are made on the client. It could be OK for the control-room applications that interact with PLCs over Modbus protocol. It is the prescribed behavior, and it is common for Modbus -- only the operator is responsible for incorrect values. However, the second group of applications are remote SCADA clients. These applications work outside of the trusted zone and if an attacker could somehow (by MitMing or by establishing a rogue connection) forge a value in the request, that could lead to really dangerous things: in some cases, to alarms or fault states.

Other weaknesses

Unprotected data storage and data integrity weaknesses. A potential security flaw appears if the application stores some data on the SD card. In certain cases, other applications or viruses could access or modify this data. Moreover, this data could be modified when the phone is connected to the PC. During our tests, we've found that some applications store sensitive data on the SD card, like logs (which could contain user names and server IPs), connection settings, or (in case of mobile HMI) even

project files. The first two cases could ease the attacker's access to the servers; the last case has been already reviewed in the application classification section. An attacker could change the project configuration, and an unsuspecting engineer will damage the process while thinking they are doing the right thing.

Absence of password protection. Are there any password protection mechanisms implemented by the mobile application? If not, an attacker who has gained control of the mobile device for a short amount of time (e. g. if engineer left an unlocked smartphone on the table for a couple of minutes), change configuration of PLC or SCADA, get important information about the industrial process, or send dangerous commands to the system. Even 4-digits password could prevent attacker from such actions. More than 70 % of reviewed apps have no password protection.

Client-side or server-side Denial of Service. Several applications crash if they receive incorrect data from the other side of the communication channel. Crash on the client side will not let the engineer view sensitive process data or send important command to the system. Crash of the server-side application could lead to more serious consequences, such as alarm or fault state of the industrial process.

Other minor weaknesses. Only half of the applications use code obfuscation, and no application detects if the tablet/smartphone was rooted.

Summary of all bugs:

Vulnerability/weakness	Control-room app	OPC-/MES- client	Remote SCADA client
M1: Weak server-side controls	0	**1**	0
M2: Insecure Data Storage	**2**	0	**3**
M3: Insufficient transport layer protection	0	**3**	**6**
M5: Poor authorization and authentication	*n/a*	**3**	**4**
M6: Broken crypto	0	**2**	**7**
M7: Client side injection	**1**	0	0
M8: Security decisions via untrusted inputs	*n/a*	0	**1**
No password protection	**4**	**5**	**4**
Denial of Service	**1**	0	**3**

5. Example vulnerabilities and attacks

In this section, some sample vulnerabilities and possible attacks based on them will be shown. Of course, we cannot disclose too many details because several vendors are now working on fixes. We will hide some details but show the mainline of such bugs.

Attack 1: triggering insufficient process parameter checks on the server side

To exploit this vector, the attacker should:

- Either be connected to the network with the target client/server (in case of control room application);
- Or know the remote control SCADA endpoint (server address) and have valid login credentials.

E. g. a potential victim uses their smartphone at a public Wi-Fi AP (e. g. in a restaraunt or trade center). If there is no encryption (or weak/vulnerable one), the attacker could MitM into the connection with the server (using, for example, ARP spoofing). When the legimate request will go to the system, it can be changed. Sometimes, it can lead to “funny” situations (see fig. 5).

Figure 5: Modified parameters

Of course physical/process controls will prevent the house from heating to that temperature. But you never know how correctly they will work.

In case of remote endpoint or isolated network, if an attacker can somehow aquire correct connection credentials, it could lead to similar or more dangerous consequences. Here is the example query (from one real application) that sets the temperature value to abnormal:

```
GET /scgi/?c8785. <removed>=26999& HTTP/1.1
Cookie: sessionid=98aec9bc19d2bed32d3cc9a1140920e2
Host: <removed>
Proxy-Connection: close
Connection: close
```

Attack 2: HMI database compromise

As we mentioned earlier, several applications stores HMI databases (circuits, interfaces, connection parameters, and HMI projects) on the SD card. Fig. 6 shows how this data may look on your smartphone.

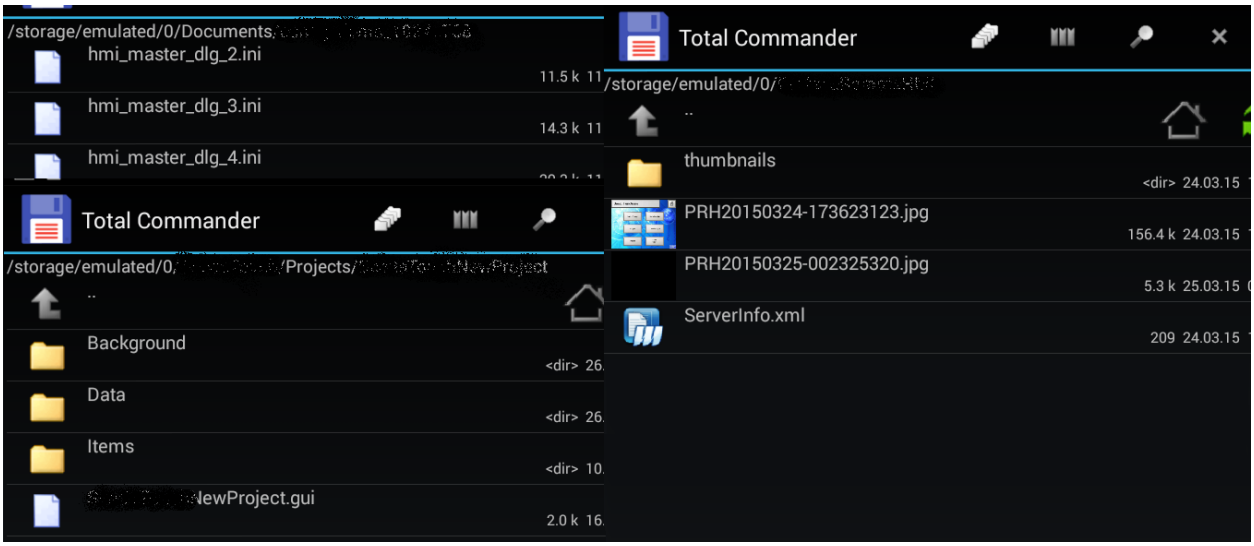

Figure 6: HMI project storage

In the picture, you can see how a real HMI project looks like. This application moves some liquid from one reservoir to another. To control this process, you have buttons (start/stop) and a flow sensor (Fig. 7).

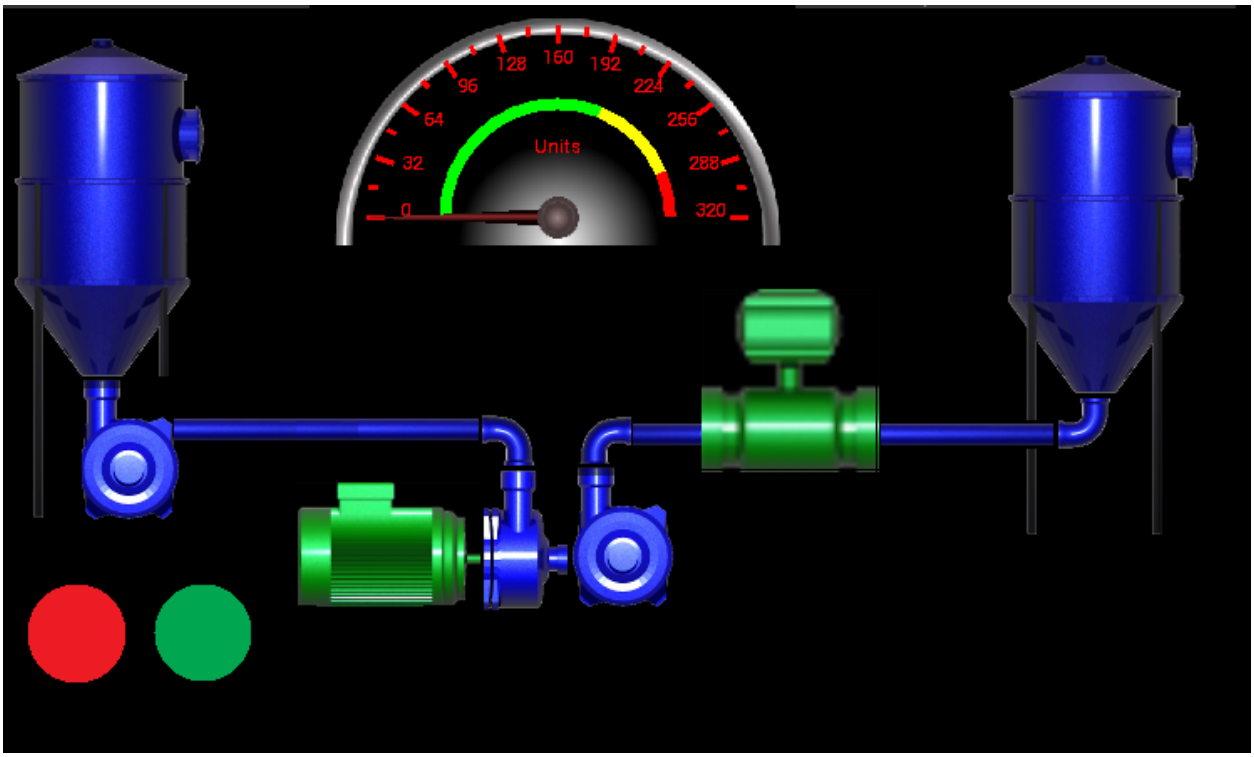

Figure 7: HMI flow sensor

If HMI project data is somehow compromised (virus, another application vulnerability, direct access to SD card from PC, etc.), the attacker can slightly reconfigure or change this interface. They could replace component events and logic or, for example, sensors data sources. When the operator launches the application next time, they could see something like Fig. 8.

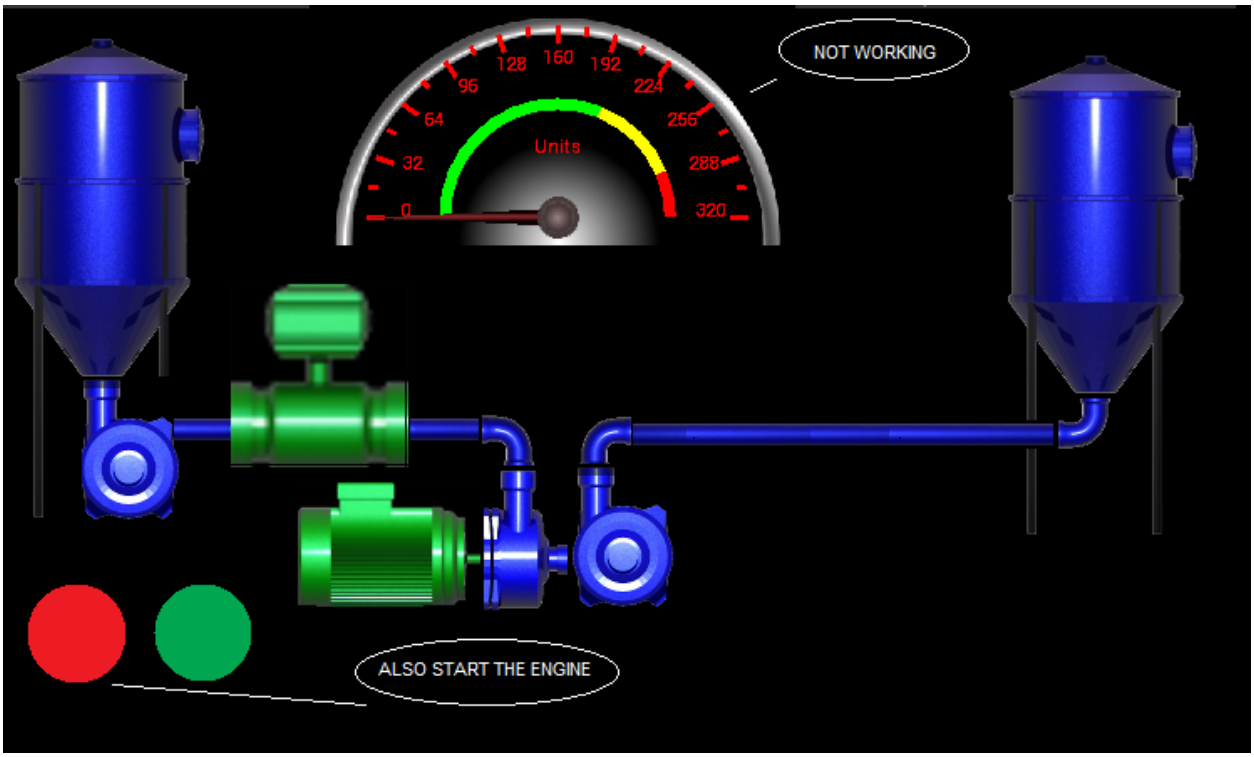

Figure 8: Modified interface

Now, BOTH buttons launch the process. Of course, you can say that physical/low-level controls will prevent the danger, but this will *at least* lead to an alarm or even fault state.

Example vulnerability 1: weak password hashing function

We've alreasy discussed weak hashing functions in mobile applications. Here is a sample:

```
String str1 = ((EditText)findViewById(2131230732)).getText().toString();
String str2 = ((EditText)findViewById(2131230733)).getText().toString();
if (str1.length() == 0)
{
  Toast.makeText(this, "Input the Username", 1).show();
  return false;
}
String str3 = Long.toString(ZipPassword(str2));
Log.i("passcode", str3);
localNetWebServiceCall.addProperty("username", str1);
localNetWebServiceCall.addProperty("password", str3);
localNetWebServiceCall.Call();


long ZipPassword(String paramString)
{
  int i = 0;
  int j = 0;
  String str = paramString.toUpperCase();
  for (int k = 0;; k++)
  {
    if (k >= str.length()) {
      return j + 65536 * i;
    }
    i = (short)(i ^ str.charAt(k));
    j = (short)(j + str.charAt(k));
  }
}
```

Maximum value result of such function is $\sum_i p_i + 65535 \bigoplus_i p_i < 2^{25}$ values. A potential attacker could bruteforce this hash fairly easily.

Example vulnerability 2: Server-side denial of service

During our analysis, we've found several denial of service vulnerabilities. Here is one sample. Potential attacker could shutdown the mobile backend and prevent operator from acessing industrial process data and controls. (fig. 9):

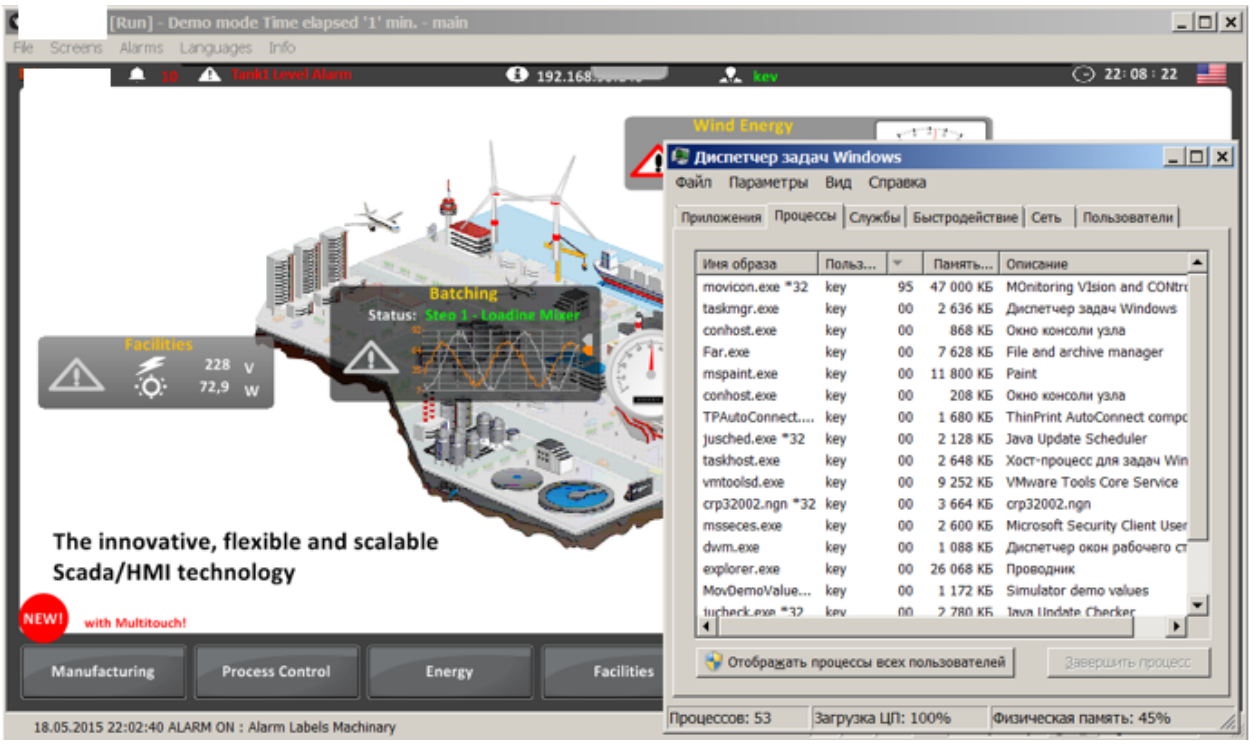

Figure 9: Server-side DoS.

6. Conclusions

During our research, we have reviewed 20 Android applications that work with the ICS infrastructure in different ways. We've examined control room applications, OPC and MES clients, and remote SCADA clients. It should be pointed out that we didn't find even one application without weaknesses and/or vulnerabilities. In short, most applications have problems with authentication and local/network data integrity. The most dangerous consequence of these vulnerabilities is “compromising” the operator themselves, i. e. give them a false understanding of current undustrial process state. We have discussed several attacks that could be conducted with the help of such vulnerabilities.

SCADA and ICS came to the mobile word recently, but they brought old approaches and weaknesses. Hopefully, due to the rapidly developing nature of mobile software, all these problems will soon be gone.

7. Thanksgiving service

This research would not be possible without the help of many people and companies, and we want to mention them. Thanks to:

- **Dmitry 'D1g1' Evdokimov** for "some binary magic" and great help in reverse-engineering ARM native code
- **Marina Krotofil** for some threat ideas
- **Alina Oprisko** for editing and help with translation of this papers
- **Alexander Popov** for the great presentation background picture