ELECTROMECHANICAL PIN CRACKING IMPLEMENTATION AND PRACTICALITY

Justin Engler – jengler[at]isecpartners[dot]com
Paul Vines – pvines[at]isecpartners[dot]com

iSEC Partners, Inc.
123 Mission Street, Suite 1020
San Francisco, Ca 94105
https://www.isecpartners.com

July 8, 2013

Abstract

PINs are often used for security on devices with touch-screen or physical keypads. Though other attacks are often available for these targets, in some cases only a brute-force attack is possible. This paper discusses several approaches for physical attacks against these systems and analyzes the practicality of these approaches against common brute-force countermeasures.

1 THE PROBLEM: LOCAL PASSWORDS WITHOUT AN AVAILABLE SOFTWARE OR ELECTRONIC ATTACK.

Many devices are secured with passwords. Assuming a password cannot be observed in use or obtained from someone who knows it, possibly the best way to subvert the security of these devices is to attack them at the software level, by either:

- bypassing the password completely (via a logic flaw, a vulnerability, etc.),
- resetting the password to a known value,
- or performing a password-guessing attack without any other security or user interface restrictions (Offline brute-force)

Though a more educated password-guessing scheme might also be used (dictionaries, patterns, etc.), for the purposes of this paper we will call all password-guessing attacks "brute-force attacks".

In some cases, there will be no software-based attack possible. An example might be a mobile device's lock screen. Perhaps no jailbreak or root technique is available, or perhaps other considerations simply prohibit its use (such as forensic evidence requirements or scoping decisions on a security assessment). Other devices, like hardware PIN pads, are also common in this scenario. When software-based attacks are not feasible, the next best attack is an electronic attack:

- Extract a password or key from the memory of a device via a chip reader or cold-boot attack.
- Attach a device that emulates a keyboard or other input device and attempt brute-force.

- Electronically input password guesses into the system directly for systems where a password is transmitted down a wire, wirelessly, etc.

In some situations, none of the above techniques will be available. Usually this occurs when there is no available electronic interface to the device. This could be a physical limitation such as:

- a hardened device housing that cannot be defeated in the time available,
- a fully-mechanical device, or
- a device with electronic interfaces that are inactive or otherwise useless for password input

Administrative reasons (evidence requirements, scoping issues, etc.) are also often to blame. If all other avenues have been exhausted, the only solution left is to start pushing buttons.

There are a variety of factors that might prevent an online brute-force attack against a device's interface:

- The device is reset after a number of incorrect guesses
- The device has a long, complex password that would be time-prohibitive to guess exhaustively
- The device locks out the input after incorrect guesses
- There is insufficient time to brute-force the device
- The labor cost to brute-force the device is prohibitive

Devices with the first two defenses will be relatively safe from the techniques described later in this paper. The last three can all be subverted to some extent, and the remainder of this paper will discuss techniques to perform an online brute-force attack in these situations. Because we are assuming that the passwords will be relatively easy to guess, the rest of this paper will refer to PINs instead of passwords. The techniques would still work with more complex alphanumeric passwords, but issues of scale arise.

2 SOLUTION: LOCAL ELECTROMECHANICAL BRUTE FORCE

We need a way to actuate the buttons or touch screen on a device to facilitate rapid brute-force attacks without having a user manually press the buttons. There are several electromechanical means to achieve this, which we will call Electromechanical Brute Force Systems (EBFS), but in all cases control software will be required.

2.1 CONTROL SOFTWARE:

Control software for electromechanical brute-forcers must be capable of the following:

- instructing the device to push a single button or a series of buttons.
- recognizing when a PIN was successfully entered.
- Configuring or calibrating any attached electromechanical brute-forcers attached (either automatically or with user assistance).
- generating random or sequential PINs to guess.
- loading a predefined list of PINs to guess.

A program, BruteController, was written in Python to achieve these goals. BruteController can detect an attached EBFS and has a simple interface that guides the user through any required calibration. BruteController also interfaces to an attached camera to assist with calibration and to detect if a device was unlocked successfully. BruteController communicates with an EBFS via a serial port (usually emulated via USB).

The integration of the camera into BruteController is accomplished through the use of the OpenCV library. This allows the system to compensate for different device orientations and different camera angles by use of some edge detection and perspective manipulation algorithms. If the resolution of the camera is high enough (>= 4MP) and the device interface has decent contrast (standard lock screen layout with bordered buttons) the vision algorithms can also predefine regions as buttons to reduce the user's setup time. Additionally, the vision software continually observes the device interface to determine if a large change (indicative of an unlock) has occurred.

2.2 R2B2 - Robotic Reconfigurable Button Basher

R2B2 is an EBFS designed to rapidly press sequences of buttons on a touch screen device. R2B2 is derived from the delta family of robots. Delta robots were first used in the 1980s for industrial applications requiring precision and high speed with limited degrees of freedom. A typical delta robot design consists of an effector head attached to 3 sets of arms with rod end bearings or a similar mechanism. The arms are driven by 3 servo or stepper motors (sometimes 3 linear actuators are used instead). Each of the motors can be adjusted to an independent angle, and the combination of the 3 angles can maneuver the effector head in the X, Y and Z directions. The calculation required to convert a desired effector position into inputs for the various motors is called inverse kinematics (IK) and the IK required for a delta robot is some fairly basic trigonometry related to the paralellograms formed by the arms of the robot and the angles of the motors. A 3-drive delta robot cannot rotate its effector head, but some delta robots with more motors and arms can be used to provide rotation as well. The typical use of a delta robot is to hang the robot upside-down over the working area from a frame. Whatever tool the robot will use is attached to the effector head, and the head is moved rapidly from position to position. It turns out that this is ideal for rapidly pressing buttons.

R2B2 is more specifically based on the designs and firmware of Dan Royer at marginallyclever.com[1]. Dan's delta robot appears to have been designed for manufacturing, as it supports G-code, a protocol often used for 3D printers, CNC routers, and similar devices. The robot uses hobby servos commonly used in radio controlled cars and is driven by an Arduino Uno or compatible microcontroller.

In converting Dan's delta design into R2B2, a few changes were made.

- Code was added to the Arduino code to handle several new functions:
    - Move from A to B as fast as possible (existing code had to worry about movement rates and pathing, which is important for computer-based manufacturing, but only serves to slow down our application)
    - Remember a particular point and associate it with a mnemonic (this allows us to store button locations)
    - Move the effector down quickly, then back up to its previous location
    - Move the effector to a particular saved point

[1] https://github.com/i-make-robots/Delta-Robot/wiki

- Push a button (This combines the “moved to a saved point” function with the “move down and up” function)
- Some bug fixes related to the requested vs. actual position of the effector

- Effector head
  - A stylus head was added to the effector head to allow the robot to interact with capacitive screens (the type of screen most common in modern touch screen devices). A grounding wire was added to connect the stylus to the Arduino's ground contact to provide the capacitance needed to actuate the touch screen. This stylus can also press physical buttons.
- Frame
  - A frame was constructed to hold R2B2 above the working area where the device to be brute-forced will be placed.
  - Other frames are under consideration to allow R2B2 to be mounted horizontally, for use against horizontal devices (doors or automobiles with keypads, etc.).

R2B2 can potentially reach speeds of 5 touches per second or more on a touch screen device (at the time of this writing, we are limited to 2 touches per second due to a firmware bug). This is sufficient to enter a single 4-digit PIN every second, which is roughly equivalent to a human quickly entering a PIN. R2B2 is also easily capable of working with physical devices such as keypads or even mechanical push-button locks, as long as a method can be devised for the robot to determine if the PIN was successful via mechanical manipulation or camera observation. This requirement might be waived if the device simply needs to be unlocked and the user doesn't need to know the actual PIN used.

There is nothing magical about this particular software and hardware configuration. The R2B2 firmware will operate with other other delta robots if they are driven by hobby servos after minor changes to constants that define the lengths of the various mechanical parts of the robot. Another robot that is likely to work well with this setup is Jason Huggins' Tapsterbot[2] design.

2.3 C3BO - Capacitive Cartesian Coordinate Brute-force Overlay

Even on devices with no available electronic interfaces to allow for attacks, the input device itself may be used. A capacitive touch screen can be activated by pulling a contact to ground or a capacitor at a given location. By use of relays and wires or electrodes attached to the screen at each button, a microcontroller can electromechanically actuate each button without a moving robot.

C3BO is principally a DEF CON 20 badge with some relays attached. C3BO's speed is only limited by the speed of the relays chosen or the input speed of the touch screen device itself, and can theoretically operate much faster than a human or a physical robot. This advantage is somewhat diminished by the need to check for a screen change indicative of a successful unlock.

[2] http://bitbeam.org/2013/05/02/a-robot-on-every-desk/

3 Conclusions on the Effectiveness of Local Electromechanical Attacks

A device protected by a mechanism that erases or permanently locks the device is not a good candidate for R2B2 or C3BO unless the number of guesses required for a lock is very high. Usually a manual tester will be able to manually test a set of likely guesses with a small chance of success. Most major mobile device platforms provide this as an option, but do not enable it by default. At least one hotel safe manufacturer appears to use a permanent (or at least very long) lockout.

A device protected by a timed lockout might be resistant to R2B2 and C3BO, but this depends on the details of the lockout. Apple's iOS devices by default provide a scaled lockout, where the user must wait for a longer period with each wrong guess. This is highly effective against brute-forcing techniques, as the wait time required quickly becomes intractable. Google's Android devices by default require a 30-second "cooldown" period after every 5 incorrect guesses. Although this might prevent casual manual brute-forcing, it does not provide a sufficient barrier to R2B2 or C3BO in many cases. A 4-digit PIN (the default if pattern locking is not used) can be fully exhausted in approximately 20 hours assuming 1 PIN entry per second and a 30 second cooldown every 5 guesses. Most real world PINs will take significantly less time.

Overall, these attacks are effective, but only in very limited circumstances.

PIN character set and length	*1 PIN per second*	*1 PIN per second, plus 30 seconds every 5 guesses*
3 Digits	16 Minutes	117 Minutes
4 Digits	167 Minutes	19.4 Hours
5 Digits	27 Hours	8.1 Days
6 Digits	11.8 Days	81 Days
4 Lowercase + Digits	19.4 Days	136 Days
7 Lowercase + Digits	2484 Years	7.83e10 Centuries
4 Printable ASCII (94)	2.48 Years	7.81e7 Centuries
7 Printable ASCII (94)	20563 Centuries	6.48e13 Centuries

Table 1: Time to exhaust a PIN's combinations

4 FUTURE WORK

BruteController and R2B2 are easily extensible to allow brute-forcing of non-numeric passwords using the whole keyboard. Exhaustive testing is completely intractable, but a targeted wordlist could prove practical. C3BO would require a separate wire for each character, or a grid system that can actuate points at the intersections of the grid.

Because of its programmability, R2B2 might be useful for fuzzing of physical interfaces. UI vulnerabilities such as lock screen bypasses, which are usually a sequence of UI inputs performed in rapid succession. This might require separate actuators to press hardware buttons that are difficult to reach from the front side of the device.