2018 Embedded Capture-the-Flag Challenge:

Secure ATM with Chip-and-PIN

Contents

1 Challenge Overview

You are part of the design team tasked with implementing a modern chip-and-PIN ATM system for your newest customer: a large banking chain. The bank has contracted an outside firm to design the user interface of the ATM and wants to maintain the existing bank administration software they currently use, so your design will have to use pre-defined application programming interfaces (APIs). All other implementation decisions are up to you.

Your challenge is to design and implement the firmware, software, and protocols for the ATM card, the ATM, and the Bank Server to support secure cash withdrawals.

Figure 1: Bank-issued Smart Card

The biggest consideration of the entire system (beyond just working correctly and dispensing money to users), must be security. Can you imagine if someone was able to make counterfeit banking cards by secretly inserting fake card-readers and cameras into ATM locations[1]? Or worse, someone could calculate the secret cryptographic keys for all of your banking cards because of an error in its implementation[2]? Previous MITRE eCTFs have shown that securing hardware is harder than it may seem. Even with extensive security reviews, it's easy to miss important vulnerabilities. And of course, you're in a hurry to get your product out to market!

There are several threats that the bank has identified and wants to protect against, including:

- Criminals may try to create a "clone" of a bank card after learning the associated PIN number, allowing them to steal money from the account
- Criminals may try to extract PIN numbers from stolen bank cards in order to steal money from the account
- Hackers might try to extract money from the ATM without a bank account or from a bank account that doesn't have sufficient funds
- Hackers may try to exploit a bank server to access user account information

Your system must meet a set of requirements (specified throughout this document) and defend against as many attacks as you and the other teams can think of. You must design and implement a working banking system which includes support tools for the essential functions of provisioning new ATMs and Cards, opening new bank accounts, and changing Personal Identification Numbers (PINs). Once your system is completed, it will be subjected to attacks from the opposing teams, while you get a chance to attack the designs from the other teams. *The purpose of this scenario is to encourage a focus on security for the embedded system and to allow all types of attacks.*

[1] https://www.justice.gov/opa/pr/four-more-members-atm-skimming-conspiracy-targeting-multiple-new-jersey-bank-locations-plead
[2] https://arstechnica.com/information-technology/2017/10/crypto-failure-cripples-millions-of-high-security-keys-750k-estonian-ids/

1.1 Competition Phases

This is an attack-and-defend capture-the-flag, so there are both offensive AND defensive phases, as shown below.

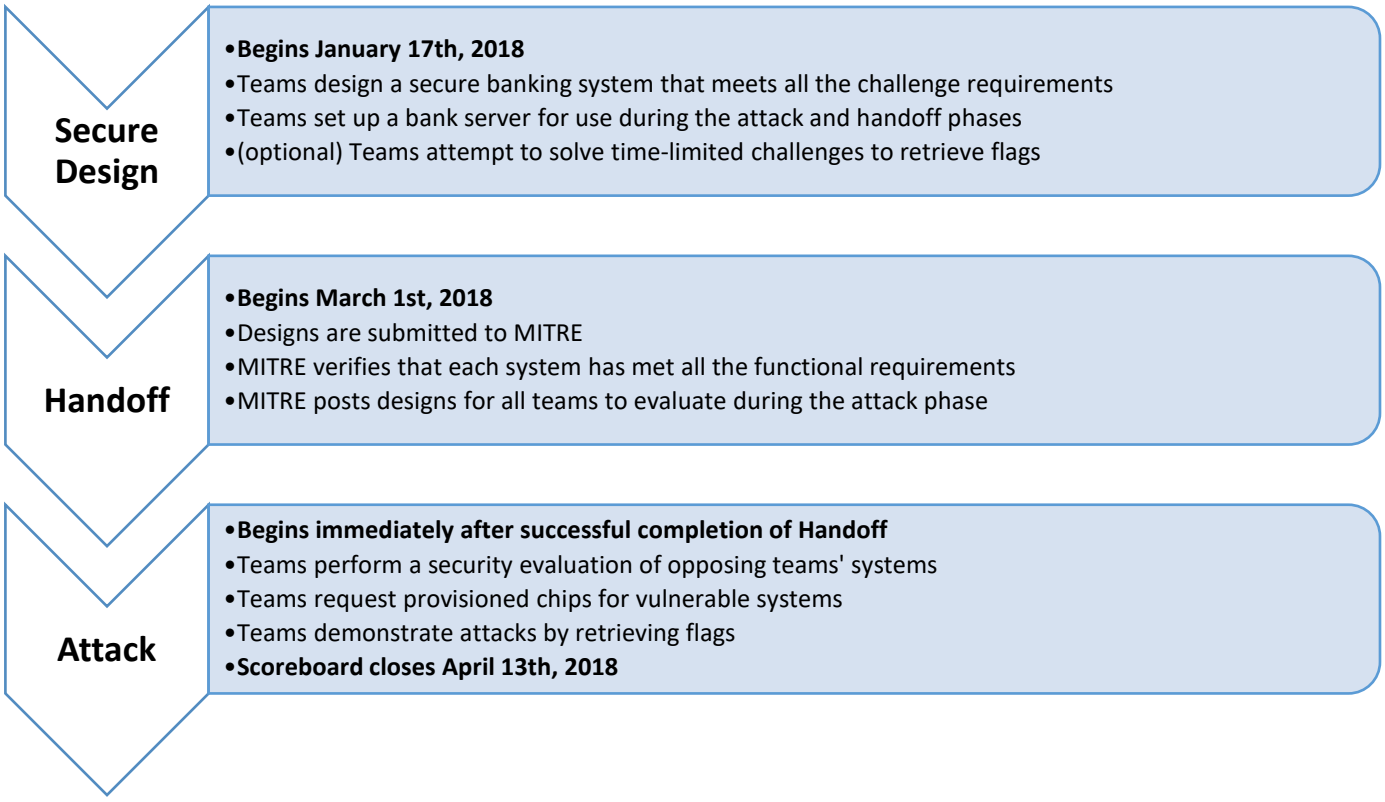

Figure 2. eCTF Phases

For each team, the Handoff phase is completed when the team submits a system that passes MITRE's requirement tests. Therefore, the date and time of transition between phases may vary between teams. Each team is allowed into the Attack phase by the eCTF organizers as soon as that team completes Handoff. For example: if Team-A and Team-B both submit systems on the handoff date, but only Team-A's system passes the tests, then Team-A will move to the Attack phase, while Team-B remains in the Handoff phase until they resubmit their system with the necessary fixes to pass the tests.

There are significant scoring advantages to entering the Attack phase as soon as possible. We highly recommend setting a schedule for your design and implementation and sticking to it as closely as possible. If your schedule starts to slip, your team will need to make hard decisions about what security features can be excluded to submit a working design on time (or at least as soon as possible). Final testing always takes longer than expected and sometimes reveals tricky problems – so plan to start your final testing earlier than you think is necessary.

2 The Kit

Building a banking system from scratch would be a lot of work, so MITRE is providing a full example system (both hardware and software) that meets the functional requirements for the competition. You can use this system for testing and as a reference to help understand the requirements, and you may also use it as a base to build on for your own system.

2.1 Hardware

MITRE will provide each team with a set of hardware and peripherals required for the competition. Teams are free to obtain additional hardware to increase testing and attack development capabilities. Additionally, teams may want to look at obtaining the PSOC M-series prototyping board[3] which is similar to the competition platform, but comes with an integrated programmer. However, your system must be designed to work with the hardware provided in the Kit; switching to a completely different platform or modifying the hardware during the design phase is not allowed.

2.1.1 The Chip — Cypress PSoC4

The chip that we'll be using for this challenge is the Cypress PSoC4[4] which combines an ARM Cortex-M0 architecture with programmable digital *and* analog logic units. Check out the Datasheet for the chip[5].

2.1.2 The Development Board — PSoC 4200 Protoyping Kit (CY8CKIT-049-42xx)

The board[6] is designed to make it easy to build circuits and interface with external hardware. It can be modified and instrumented for side-channel attacks, fault injection, or any other analysis you'd like to try. We hope that the low cost of the hardware (under $5) should encourage more adventurous experimentation and attack development.

[3] http://www.cypress.com/documentation/development-kitsboards/cy8ckit-043-psoc-4-m-series-prototyping-kit
[4] http://www.cypress.com/products/32-bit-arm-cortex-m0-psoc-4
[5] http://www.cypress.com/file/302286/download [PDF]
[6] http://www.cypress.com/documentation/development-kitsboards/psoc-4-cy8ckit-049-4xxx-prototyping-kits

2.1.3 The Programmer — MiniProg3

The MiniProg3[7] is a programmer/debugger for PSoC microcontrollers. After connecting the MiniProg to the programming headers of the development board, the associated software can load compiled firmware onto the microcontroller and help debug the running code. The debugging interface is capable of setting breakpoints, stepping through code, and live viewing of memory and registers.

2.2 Example Code

Example code that satisfies all the competition requirements can be found at:

https://github.com/mitre-cyber-academy/2018-ectf-insecure-example

Be warned that this example offers **no** protection from attackers. Also, just like most code that you can find on the Internet, many security issues may exist in our example code. Obviously, these issues will persist in your system if they are not identified and removed.

Note: The scripts listed below are provided with the example code and will be used by the eCTF organizers during testing and provisioning in the Handoff Phase. **Do not modify these scripts! If modified, your system will not pass our acceptance testing.** These scripts are:

```
Build_Program.py

interfaces/
      admin_interface.py
      atm_interface.py
      provision_interface.py
```

[7] http://www.cypress.com/documentation/development-kitsboards/cy8ckit-002-psoc-miniprog3-program-and-debug-kit

3 Development Environment and Support Utilities

Setting up a development environment for programming and debugging embedded systems can be challenging. The same can be said for developing and testing docker containers for use in a cloud environment. This competition asks you to do both! To help jumpstart your development, this section offers important information and advice for setting up your own environment. We recommend reading through the entire section, but the important recommendations are summarized in Section 3.1 Quick Start Guide.

Please remember that the primary goal of this challenge is to learn about embedded security and not struggle over embedded platform issues. If you are unclear about something or having problems please contact the eCTF organizers. Most likely, other teams will be having similar issues as well and it is in everyone's interest to resolve them as quickly as possible.

3.1 Quick Start Guide

3.1.1 Suggested Setup for Windows

- Install Cygwin[8]
  - Note: This is recommended primarily for the utilities 'socat' and 'GNU make'
  - During installation, search 'socat' and 'make', find the appropriate packages and click on the toggle marked 'Skip' so that it shows a package version number, then install Cygwin as normal
- Install and run PSoC Creator and PSoC Programmer
- Use Docker Machine and VirtualBox to create Linux VMs to run Docker containers
  - Both tools can be installed through the Docker Toolbox installer (recommended)
  - Use 'Docker Quickstart Terminal' to run docker-machine and docker commands
- Open a Cygwin terminal and use 'socat' to forward local traffic on local ports 1336 and 1338 to the same ports at the IP address of the Linux VM

3.1.2 Suggested Setup for Linux

- Download VirtualBox and a Windows 10 VM
  - Install PSoC Creator and PSoC Programmer tools within
- Run Docker containers locally

3.1.3 Suggested Setup for Mac

- Download VirtualBox and a Windows 10 VM
- Install PSoC Creator and PSoC Programmer tools within
- Use Docker Machine and VirtualBox to create Linux VMs to run Docker containers
  - Both tools can be installed through the Docker Toolbox installer (recommended)
  - Use 'Docker Quickstart Terminal' to run docker-machine and docker commands
- Use 'socat' to forward local traffic on local ports 1336 and 1338 to the same ports at the IP address of the Linux VM

[8] https://www.cygwin.com/

3.2 Important Details

3.2.1 PSOC Creator IDE

The sole method to build software for the PSoC chips for this competition is the PSOC Creator IDE[9] (please use version 4.1 ONLY). Programming can be accomplished using the PSOC Creator or PSOC Programmer, which comes with the MiniProg3. Unfortunately, both tools are currently only compatible with Windows. *Note: We believe the strengths of the PSoC platform (especially the possibilities afforded by the programmable digital and analog hardware) and low cost are compelling enough to overlook this compatibility limitation.*

For students without access to Windows, Microsoft provides several free Windows 10 virtual machines (VMs) [1] [10] [2] [11] which can be used to run the Creator IDE. The free VMs are designed to expire between 30 and 90 days from the original download/installation date, therefore Microsoft suggests creating a snapshot of the initial installation so that the VM can be rolled-back to a pre-determined state. This intentionally allows for use of the VM for significantly longer than 30-90 actual (calendar) days. By using snapshots, a single VM from Microsoft should be sufficient for the entire duration of the eCTF competition. When rolling back to an old snapshot, be careful to first copy off any work that may be stored in the VM – a smart workflow might be to avoid storing any work on the VM itself.

As an alternative to using the free Microsoft VM, be sure to check with your school's IT department, as many schools have Windows VMs or full OS installations available for students.

3.2.2 Docker Containers

This competition will make use of Docker containers[12] for code distribution. Docker containers are designed to run consistently on any compatible host OS and allow for minimal distributions of code and all the supporting libraries and tools necessary to run that code. There are many online tutorials to help get started with Docker and the provided reference design code includes two examples of Dockerfiles (a configuration file that is used to build Docker containers).

So-called "native" Docker on Windows and Mac OS does not support USB/serial pass-through which will be required for this competition. As result, Docker containers will need to be run in a Linux environment. If you do not have access to a Linux host, this can be accomplished by a creating a Linux VM by using the Docker Machine[13] tool provided by Docker (which can be found in Docker Toolbox[14] and is the recommended solution for Mac and Windows). To set up USB/serial pass-through please see the following links: Linux[15] and Windows/Mac[16]. Additionally, the Docker commands found in the example code 'Makefiles' (see "GNU 'make'" below) are designed to allow USB pass-through.

3.2.3 Port Forwarding Utilities

To reduce the configuration requirements on the individual components to be developed, a fixed set of TCP/IP network addresses and ports will be used in each banking system component. For example, the banking admin interface must always listen on port 1338. Port forwarding utilities will be used where necessary to redirect TCP/IP traffic to intended recipients, such as when different components are hosted on separate computers or servers, as will be the case during the attack phase. During the handoff and attack phases the organizers will be using 'socat' (documentation[17], windows

[9] http://www.cypress.com/products/psoc-creator-integrated-design-environment-ide
[10] https://developer.microsoft.com/en-us/windows/downloads/virtual-machines
[11] https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/
[12] https://www.docker.com/what-container
[13] https://docs.docker.com/machine/
[14] https://docs.docker.com/toolbox/
[15] https://stackoverflow.com/questions/24225647/docker-any-way-to-give-access-to-host-usb-or-serial-device
[16] http://gw.tnode.com/docker/docker-machine-with-usb-support-on-windows-macos/
[17] http://www.dest-unreach.org/socat/doc/socat.html

port[18]) to forward traffic. While teams may use any port forwarding utility during the attack phase, 'socat' has the ability to record network traffic while forwarding it (a very useful feature during the development and attack phases). Here are some possibly useful examples of 'socat'

- To forward local traffic pointing at one port (say '1234') to another host and port number (IP '123.456.789.12', port '8888')
  - `socat TCP-LISTEN:1234,fork,reuseaddr TCP:123.456.789.12:8888`
- To send serial traffic from 'ttyS4' to an example TCP server (while logging all transactions)
  - `socat -lf logfile.log /dev/ttyS4,b115200,raw,echo=0 TCP:www.example.com:1337`

3.2.4 GNU 'make'

GNU 'make'[19] is a build automation tool that will be used during the competition to build Docker containers from Dockerfiles, launch the created container, and start executable code. 'make' works by adding target operations to a 'Makefile', listing requirements for and commands performed by each target. For example, the following command creates a target operation called 'build' that will create a Docker container called 'bank' if the files in 'bank_server' and 'Dockerfile' exist.

```
build: Dockerfile bank_server/*

	docker build -t bank –rm=true.
```

All testing and provisioning during HandOff and Attack phases will only use 'make' targets. Each 'Makefile' submitted must have the following targets ('logs' is optional):

- `start` – runs the Docker container and executes code within
  - Builds a target Docker container from a Dockerfile if it doesn't exist already
- `stop` – stops execution of any code running in the container
- `clean` – stops and removes the Docker container
- `logs` – (optional) adds logging information to the 'logs' directory run on the host; this will be used by the competition organizers to help teams debug design submissions errors during the Handoff Phase.

Only 'docker' commands (e.g. 'docker build', 'docker run', 'docker exec', etc.) should be necessary to run locally on the host, so do not add other commands to a Makefile, because the host may not support other commands. For further examples, see '`bank_server/Makefile`' and '`atm_backend/Makefile`' in the example code. Please note that 'make' is notoriously picky about whitespace. For example, all command lines within a target must be indented with a single tab, not spaces.

3.2.5 XML-RPC

XML-RPC[20] is a configurable HTTP protocol used to access functions being provided by a "remote" server. This competition uses two fixed XML-RPC interfaces so that a set of common command-line scripts written by the eCTF organizers can communicate with designs written by any team in any language. Examples of XML-RPC servers written in python that adhere to the competition specification can be found in '`atm_backend/atm_backend/___main__.py`' and '`bank_server/bank_server/__main__.py`' within the provided example code.

[18] https://www.nikhef.nl/~janjust/socat/
[19] https://en.wikipedia.org/wiki/Make_(software)
[20] https://en.wikipedia.org/wiki/XML-RPC

4 Secure Design Phase

Each team will build a secure banking system during the secure design phase. To accomplish this, teams will need to design a Bank Server, an ATM machine w/ associated Hardware Security Module (HSM), and an ATM Card. The Bank server is used by the bank administrators (i.e., eCTF organizers) to manage bank accounts and to communicate with multiple ATMs. The ATM is used by banking customers to check account details, withdraw money, and manage their ATM cards. The ATM card is used to validate to the bank and ATM that a given customer is legitimate, which is accomplished by its presence and a user provided PIN access code. While this system looks complex at first, the basic operation is fairly straightforward. Please use the example code to further understand the descriptions in the following sections. Where appropriate, specific files in the example code will be called out for deeper examination.

The following figures capture the architecture of the overall system across different views. Figure 3 shows the logical organization of the system highlighting the separation of roles and general lines of communication. Figure 4 shows the same system architecture, but is organized by the physical platforms that will host each system component and provides more specific details on each interface. Finally, Figure 5 highlights which components and interfaces are acceptable to attack.

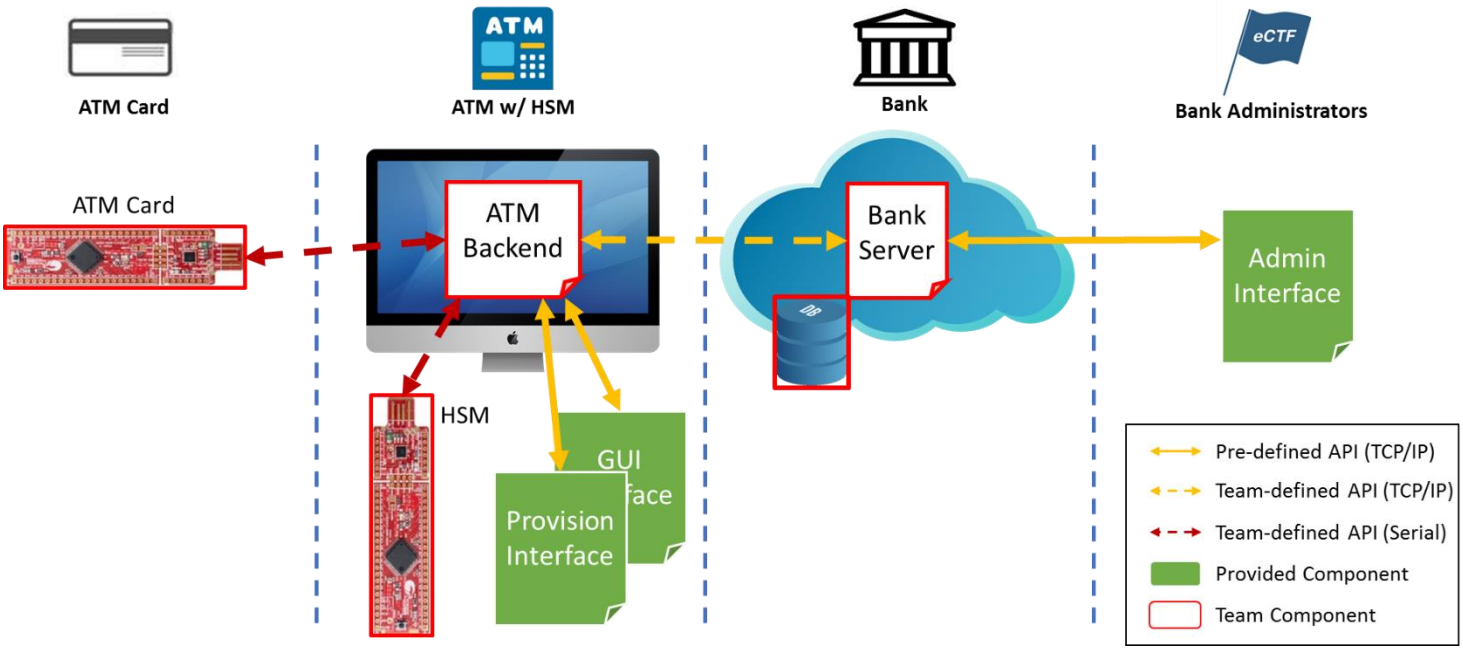

Figure 3: Banking System Architecture

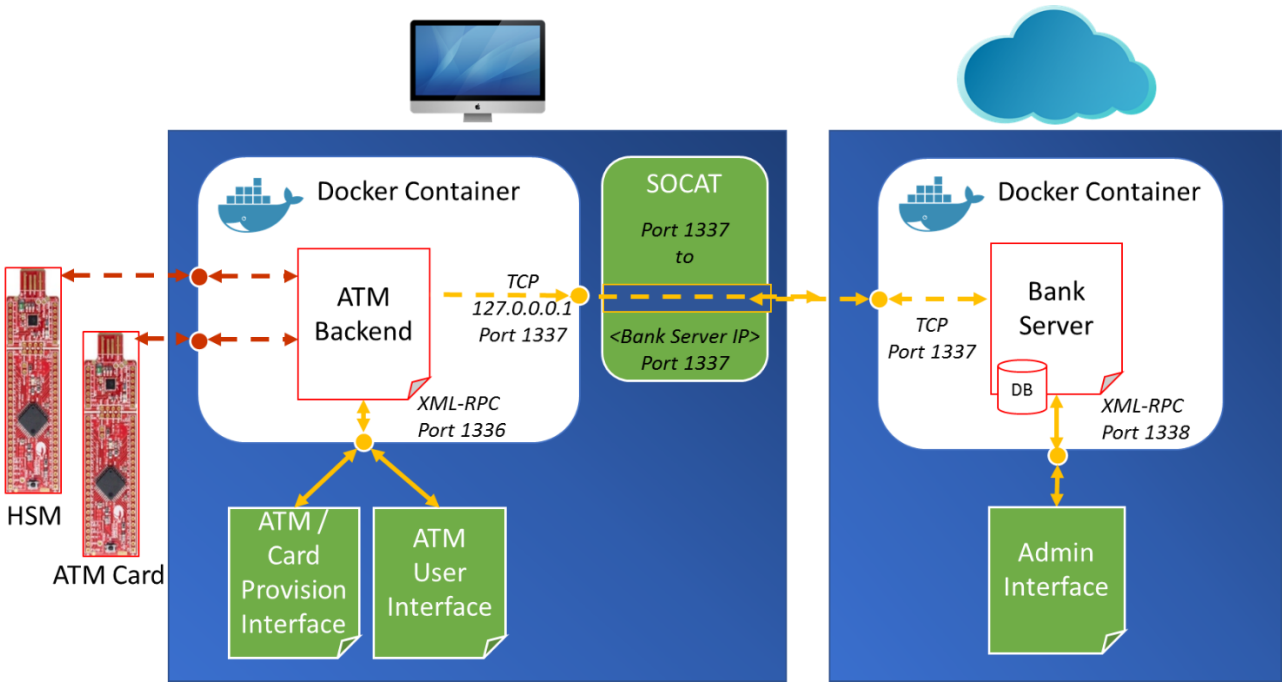

Figure 4: Banking System Architecture – Computer Layout

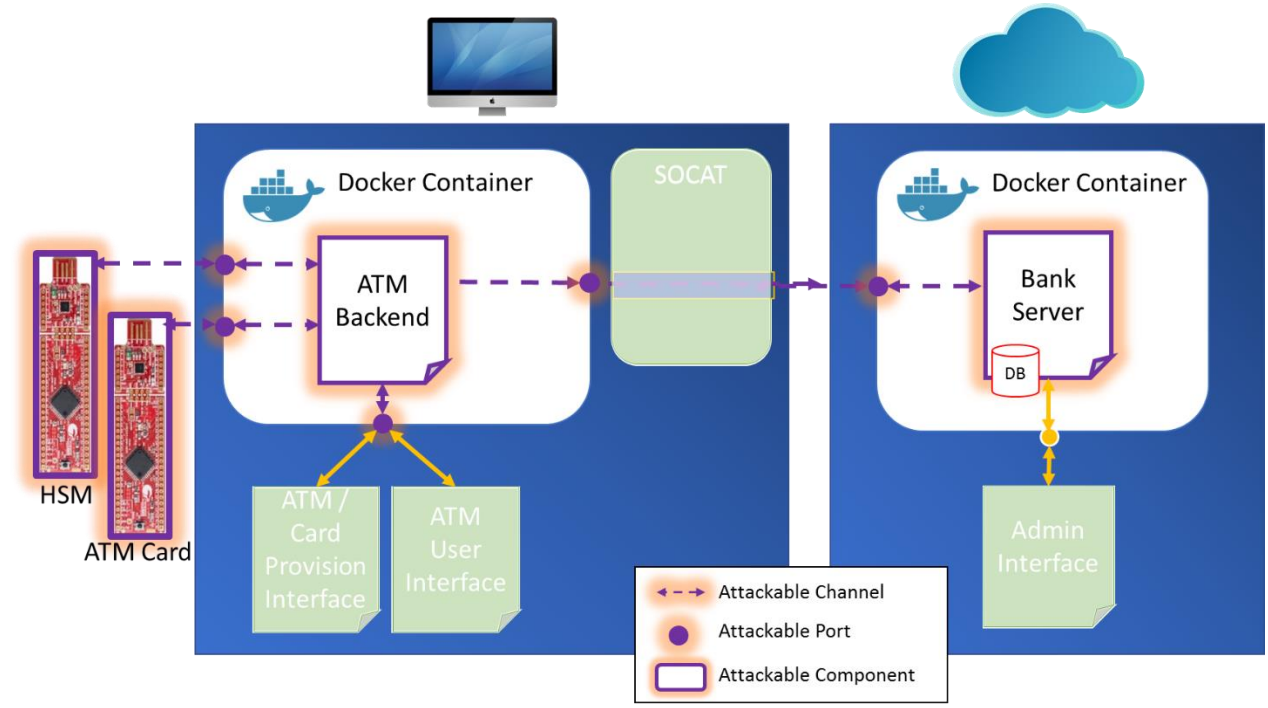

Figure 5: Banking System Architecture – Allowable Attack Phase Targets

In addition to the system's normal operation, teams will need to follow a prescribed set of steps to provision each component. These steps represent a standard set of operations that banking admins will use to start their bank server, set-up a new ATM, generate new customer accounts, and set-up new ATM cards to along with an account.

While there is some required functionality that must be present and some interfaces that must be adhered to, most of the actual implementation details are up to the team. This includes the communication protocols and algorithms used between the Bank and ATMs and between the ATM and ATM cards, the means and location of stored data, and all security measures. The only details that are not up to the teams represent actions performed by administrators or customers.

4.1 Security Goals

4.1.1 Two-Factor Authentication

The bank and ATM should be protected against bad actors through use of a two-factor authentication scheme: something you have (the ATM card itself) and something you know (the secret PIN access code). It should not be possible to withdraw money from an ATM with only an ATM card or only knowledge of a PIN. If either of those requirements is not met, money could be stolen from customer accounts. Additionally, it should not be possible to clone an ATM card, which could gain access to an account with a known PIN, but without the real ATM card.

4.1.2 Secrecy and Authenticity

Account information transmitted between the bank, ATMs, and ATM cards must kept secret from attackers that intercept this traffic. User account balances are privileged information, and exposure of this information would be damaging to the customer. Additionally, attackers should not be able to modify banking requests in transit via man-in-the-middle (MITM) attacks, nor should they be able to generate fake banking traffic that is accepted by any banking component. Should either of these attacks be possible, customer accounts could be drained of money or granted illegitimate funds at the expense of the bank.

4.1.3 Software Assurance

The ATM software is ultimately responsible for all money stored in the ATM and must be protected against attacks. That means stored bills should only be dispensed from authenticated customers and ATMs should not dispense more money that that user has in their account.

4.1.4 PIN Secrecy

PIN values used to access the ATM should never be revealed, even under extreme attack circumstances. Because customers often reuse PIN values, the bank must protect them from exposure. Specific attack scenarios that must be accounted for in this competition are attackers with physical access to a “borrowed” ATM card and a banking server data breach where all computer files (including all account information) have been stolen.

4.2 Bank Server

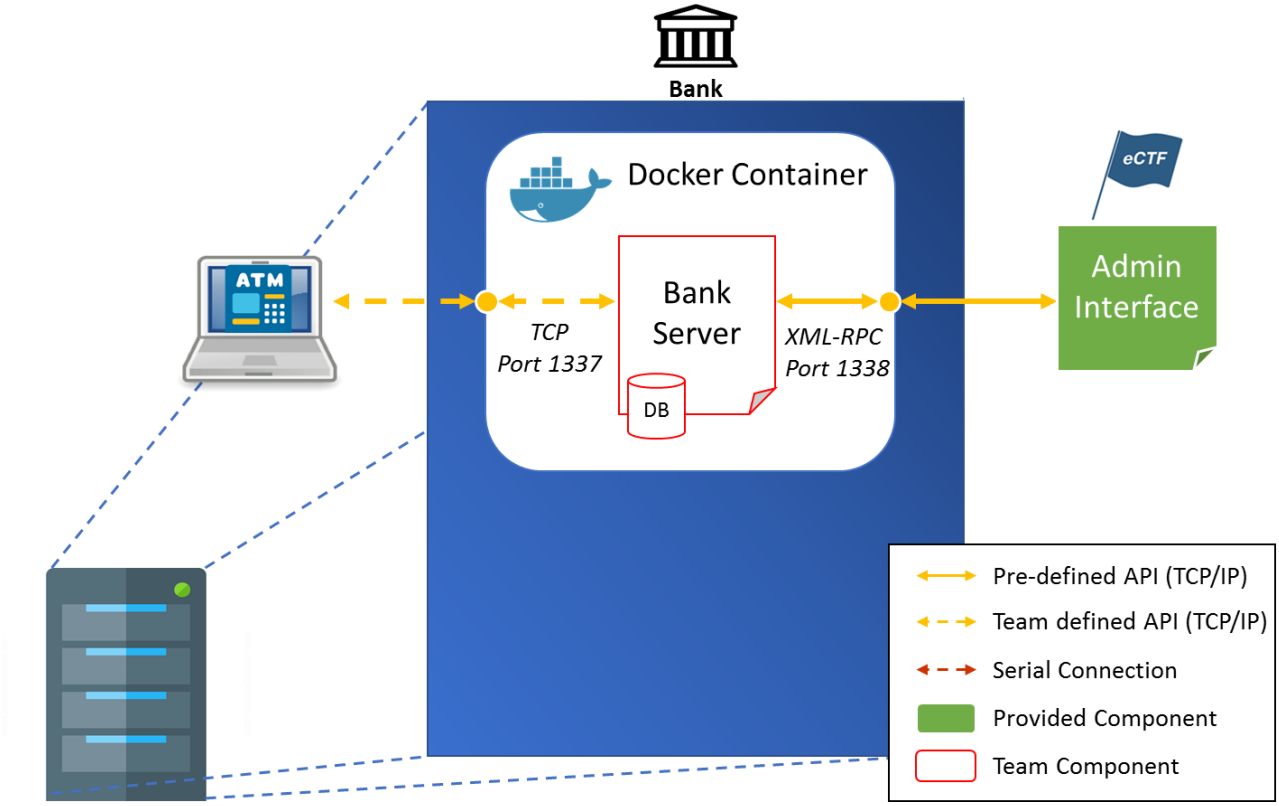

Figure 6: Bank Server Architecture

Each team will need to design a system to store banking account information and a server daemon that banking personnel can use to administer banking accounts. Necessary administrative functions consist of:

- Creating new accounts with a starting balance
- Updating the balance of an existing account, either positively or negatively
- Checking account balance for any account
- Providing material to help provision ATMs and ATM Cards (see Provisioning section below)

The ATM backend server must run in a Docker container that is compatible with native Docker and Docker Machine. The Docker container must be submitted as a Dockerfile and a Makefile (see section 3 above for more details). The first time the server is run, it should generate any extra files or material it needs that was not provided with the submission (e.g. create random cryptographic keys). The server can store files locally within the Docker container to help store account information.

4.2.1 Bank to ATM Communication

The bank server is expected to communicate with ATMs to support normal ATM operations. For example, this communication could be used to ensure that ATMs do not allow customers to withdraw more money than an account holds. This communication must be conducted via a team-designed protocol over a TCP connection. The bank server must establish a TCP server that listens for connections on port 1337.

The provided example code used an XML-RPC interface to communicate between the bank and ATMs, but other protocols are encouraged. See files '`bank_server/bank_server/bank.py`' and '`atm_backend/atm_backend/bank.py`' for communication implementation details.

4.2.2 Pre-Defined Bank Administration Interface

The bank server must support an XML-RPC interface for bank administration. The bank server must listen on port 1338 and must support the methods listed in the following table (all listed types are XML-RPC standards or standard types with additional restrictions described below):

Function	**Arguments**	**Returns**
`create_account`	• `account_name (string)` • `amount (int)`	Success: ATM Card provisioning blob `(provision base64)` Failure: `0 (bool)`
`update_balance`	• `account_name (string)` • `amount (int)`	Success: `1 (bool)` Failure: `0 (bool)`
`check_balance`	• `account_name (string)`	Success: account balance (`int`) Failure: `0 (bool)`
`create_atm`	• `provision base64`: arbitrary `base64` < 16,384 bytes used in provisioning	Success: ATM provisioning blob `(provision base64)` Failure: `0 (bool)`
`ready_for_atm`	`None`	When the bank is ready for ATM(s) to connect to its bank interface on port 1337: `1 (bool)` Otherwise: `0 (bool)`

An Admin Interface GUI can be found in the provided example code: '`interfaces/admin_interface.py`'. Any Bank Server must be compatible with this GUI.

4.2.3 Other requirements

- The Bank must be capable of tracking at least 65,536 customer accounts
    a. Accounts must contain at least the customer name and the balance
    b. Customer names up to 1024 characters must be supported
    c. Bank administrators will not use duplicate customer names during the competition
- Balances are whole integer values only
- The Bank must support up to 65,536 ATMs.
- Attacks against the bank server will only be permitted on the team-designed protocol over the TCP link. All other ports and protocols (including the Admin Interface) are out of bounds of the competition. Teams found attacking other channels will be disqualified.

4.3 ATM

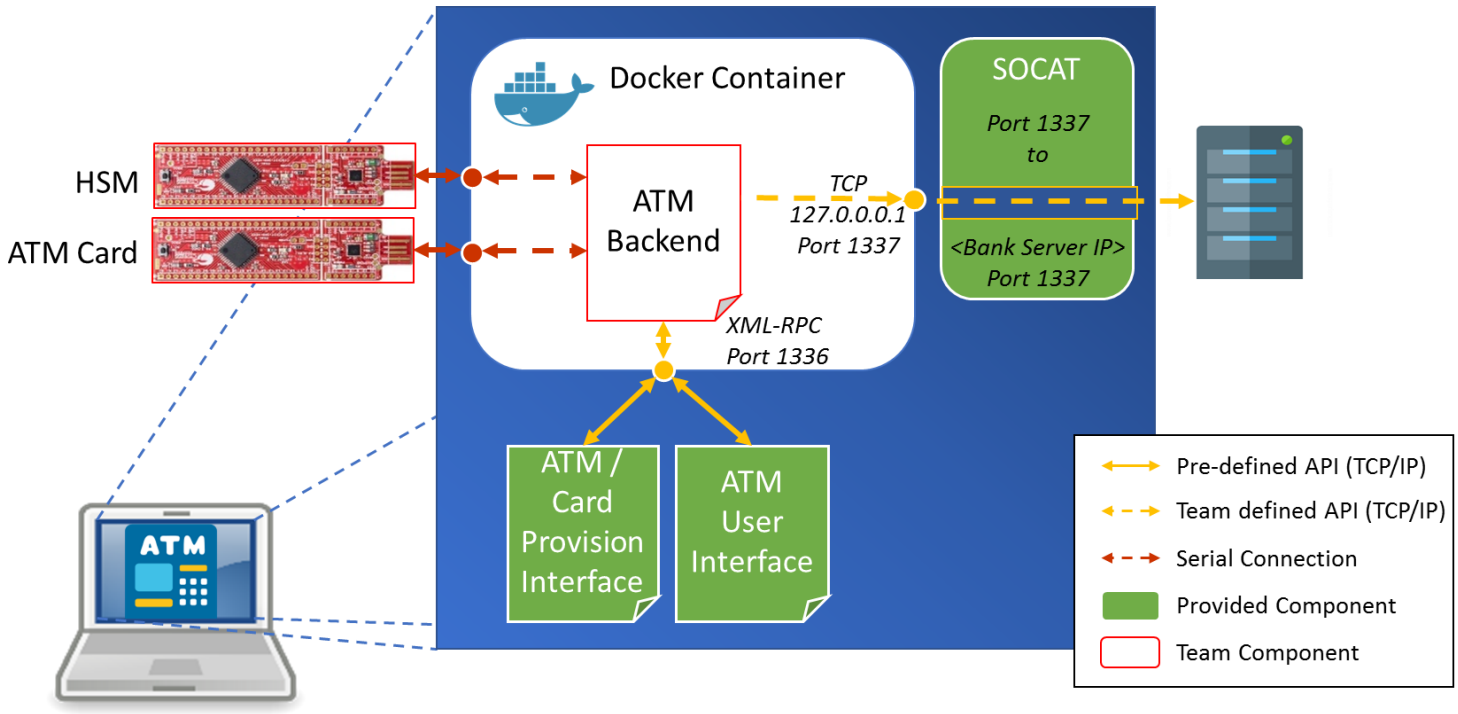

Figure 7: ATM Architecture

Each team will need to design an ATM backend server daemon that communicates directly with the Bank Server, the HSM, and ATM Cards. The ATM must support the following actions:

- Accept provisioning material to help establish connection with the Bank Server (see Provisioning below)
- Accept bills during provisioning
  - **Note**: Bills must be stored on the ATM (or HSM). Bills cannot be transmitted and stored on the Bank Server. If Bills are encrypted (including with one-time-pad) the decryption key should be treated as a part of the Bill and therefore must also be stored on the ATM or HSM.
- Provision new ATM Cards with provisioning material provided from the Bank Server (see Provisioning below)
- Authenticate customers by validating PINs and ATM cards
- Dispense bills or display account balance to authenticated users with sufficient funds
- Change PINs on attached ATM Cards

The ATM backend server must run in a Docker container that is compatible with native Docker and Docker Machine. The Docker container must be submitted as a Dockerfile and a Makefile (see section 3 above for more details). The ATM backend server can store files locally within the Docker container if desired.

The HSM can be programmed to perform any functions that teams see fit, including any secure storage or computation necessary to ensure that security goals are met. Communication with the HSM occurs over a serial protocol designed by the teams. If the HSM is removed, the ATM is not required to continue operation.

ATM Cards will also communicate using a team-designed serial protocol. Withdrawal of money, finding the current account balance, and changing the PIN should not be possible without a connected ATM Card; implementation details of this protection are up to the individual teams. Only one ATM Card is expected to be attached to the ATM at once. An

ATM Card must work with any ATM, not just the ATM used to provision it. The ATM should be able to automatically detect when a new ATM Card has been attached, allowing a customer to then authenticate themselves.

4.3.1 ATM to Bank Communication

The ATM backend is expected to communicate with the bank to support normal ATM operations. This communication, the reciprocal of the communication described in '*Bank Server: Bank to ATM Communication*' above, must be conducted via the same team-designed TCP protocol. The ATM must open a socket to the following host and port:

- IP address: 127.0.0.1
- Port: 1337

When an ATM is attempting to access a Bank Server hosted on another machine (and thus with a different IP address), 'socat' can be used to forward the TCP traffic to **Bank Server IP Address** using the following command:

```
socat TCP-LISTEN:1337,fork,reuseaddr TCP:<Bank Server IP Address>:1337
```

4.3.2 Pre-Defined ATM Provisioning and User Interface

The ATM backend server must support an XML-RPC interface for provisioning (of the ATM and ATM cards; see '*Provisioning*' section below) and normal customer operations. The server must listen on port 1336 and support the methods listed in the following table (all listed types are XML-RPC standards or standard types with additional restrictions or structures described below).

Function	**Arguments**	**Returns**
`provision_atm`	• `atm_blob (provision base64)` • `bills (bill array)`	Success: `1 (bool)` Failure: `0 (bool)`
`provision_card`	• `card_blob (provision base64)` • `new_pin(pin string)`	Success: `1 (bool)` Failure: `0 (bool)`
`withdraw`	• `pin (pin string)` • `amount (int)`	Success: requested bills (`bill array`) Failure: `0 (bool)`
`check_balance`	• `pin (pin string)`	Success: account balance (`int`) Failure: `0 (bool)`
`change_pin`	• `old_pin (pin string)` • `new_pin (pin string)`	Success: `1 (bool)` Failure: `0 (bool)`
`ready_for_hsm`	None	When ready and waiting for HSM to connect: `1 (bool)` Otherwise: `0 (bool)`
`hsm_connected`	None	After the HSM has connected and synced with the ATM (it has been identified as an HSM): `1 (bool)` Otherwise: `0 (bool)`
`card_connected`	None	After an ATM card is connected and synced with the ATM (it has been identified as a card): `1 (bool)` Otherwise: `0 (bool)`

Custom type restrictions:

- `provision base64`: arbitrary `base64` < 16,384 bytes used in provisioning
- `pin string`: an 8-character string consisting of only digits, 0-9

- `bill array`: an array of strings, each representing a `bill string`
- `bill string`: a 16-character string representing a unique identifier

ATM provisioning and customer ATM command-line interfaces can be found in the provided example code: '`interfaces/provision_interface.py`' and '`interfaces/atm_interface.py`' respectively. These tools communicate over XML-RPC using the interface described above, therefore all ATMs must be compatible with these tools and can be used to validate teams' implementations. Note: these files should not be modified.

4.3.3 Other requirements

- The ATM must be capable of storing at least 128 bills
    a. Each bill represents a single dollar
    b. Each bill is a unique 16-character string
    c. Bills must be dispensed in the order they were loaded (first in, first out)
    d. Bills must be stored on the ATM or HSM – they cannot be stored in the Bank or encrypted with a key that only the Bank has. This requirement is intended to mirror the real-world scenario in which the attacker has physical access to the ATM, which is ultimately responsible for deciding to dispense money.
- Balances are whole integer values only
- At least 65536 unique ATM Cards must be supported
- PIN values must be ascii-strings with only the numeric characters (ie. '0' – '9')
- Any function requiring a PIN should only execute if the stored PIN value matches the provided PIN

4.4 Hardware Security Module (HSM) and Bank Cards

The hardware components have no pre-defined requirements or specific implementations that must be met. They are provided to help ensure the security goals listed above.

Both components must be submitted as a Creator project file and should exist within a common Creator workspace file. In addition to these files, you must also submit all other files necessary to build and program the hardware. The workspace and projects must have the follow names and structure (see the provided reference code for an example of the layout):

```
ectf-workspace.cywrk
CARD.cydsn/
        CARD.cyprj
SECURITY_MODULE.cydsn/
        SECURITY_MODULE.cyprj
```

During acceptance testing in the HandOff phase, the firmware will be built and then programmed using the python-2.7 'Build_Program.py' script provided in the example code (and looks for the files listed directly above). Building/programming will be accomplished by running the following commands:

- `python Build_Program.py --target CARD`
- `python Build_Program.py --target ATM`

PSoC 4 supports three different protection levels[21]: 'Open', 'Protected', and 'Kill'. Please do not use 'Kill' during the design phase because it prevents the hardware from being erased and used again. **All provisioned hardware will be automatically converted to 'Kill' when provisioned by MITRE for the attack phase.**

[21] https://community.cypress.com/docs/DOC-11014

Reference designs for both the HSM and the ATM Cards can be found in the provided example code, specifically in the '`SECURITY_MODULE.cydsn`' and '`CARD.cydsn`' directories; you can load the 'ectf-workspace.cywrk' file in PSoC Creator to access both projects and associated files.

4.5 Provisioning

Before the banking system can be used by customers, a set of ordered steps will first be performed by the Bank Administrators. These steps are designed to help distribute all data and material necessary to meet the security goals listed above.

1. The Bank Server is started, followed by the ATM Backend Server
    a. Any required security material (e.g. cryptographic keys) not provided in the design submission should be generated as part of this process

2. The Admins create an ATM record at the bank

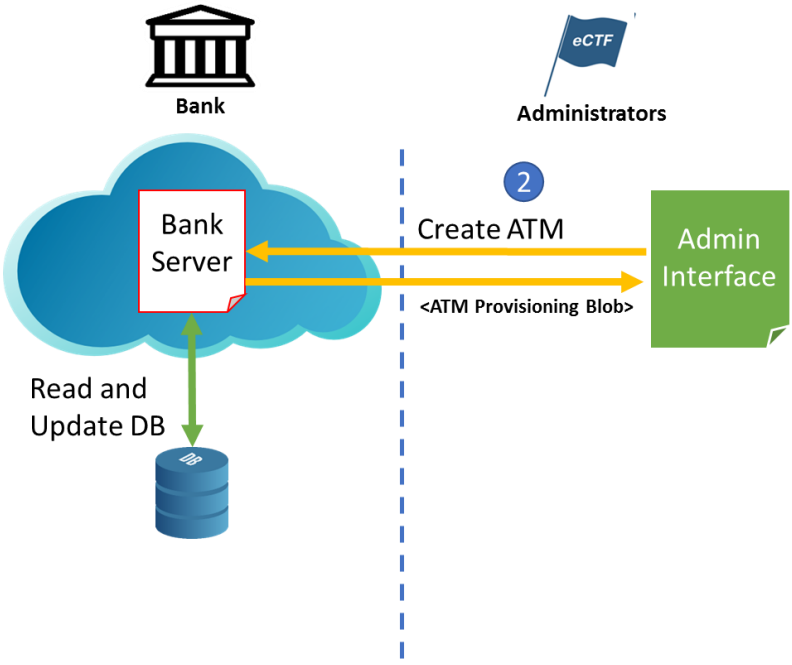

Figure 8: Admins Create ATM Record

a. Admins send the 'create_atm' command to the Bank Server over admin interface
b. (optional) the bank may generate secret material (e.g. cryptographic keys) to share with the ATM and/or update the local database to create a record of the ATM
c. The Bank Server returns an <**ATM Provisioning Blob**> which is used to provision the actual ATM in the next step

3. Admins Provision an ATM

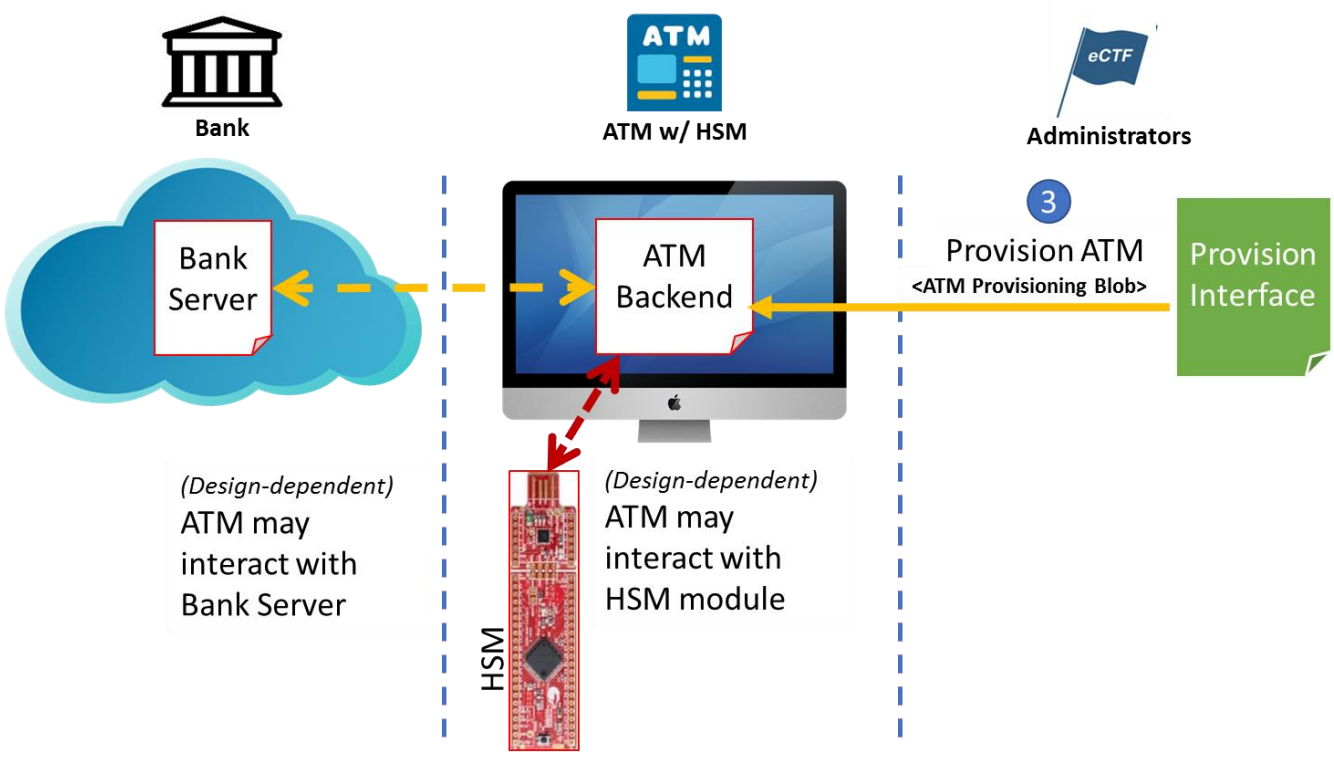

Figure 9: Admins provision new ATM

a. Admins send 'provision_atm' command to the ATM interface and supply the <**ATM Provisioning Blob**>
b. (optional) ATM may interact with Bank Server and HSM to perform additional initialization steps

4. Admins create a new customer account

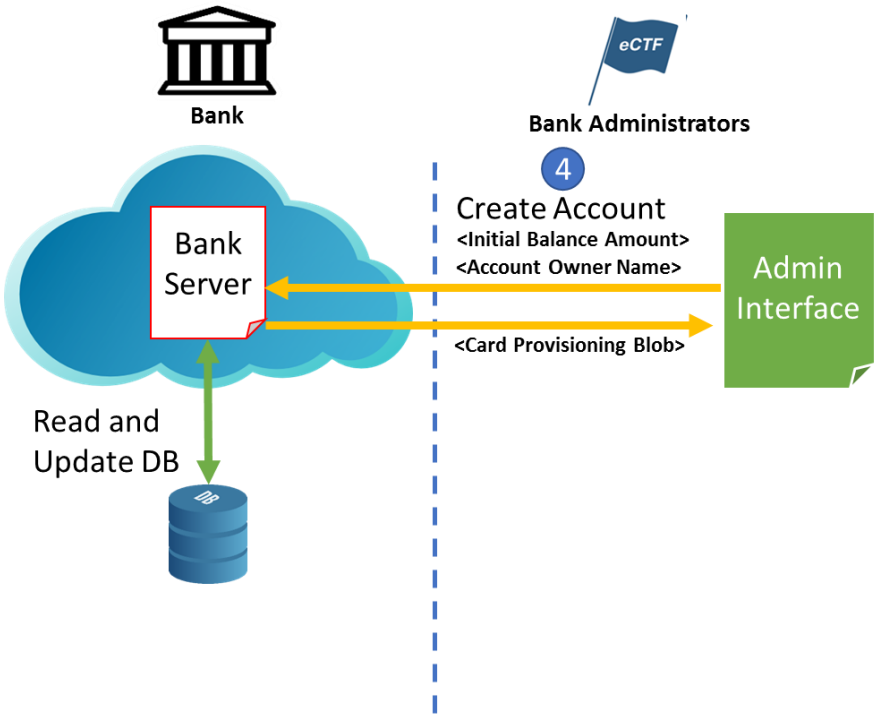

Figure 10: Admins create new customer account

   a. Admins send a "create_account" command to the Bank Server and supply an owner name.
   b. (optional) The bank may generate secret material (e.g. cryptographic keys) and/or update the local database to create a record to support the new account
   c. The bank returns a **<Card Provisioning Blob>** which is used to provision the ATM card for this account.

5. Admins provision a new ATM Card

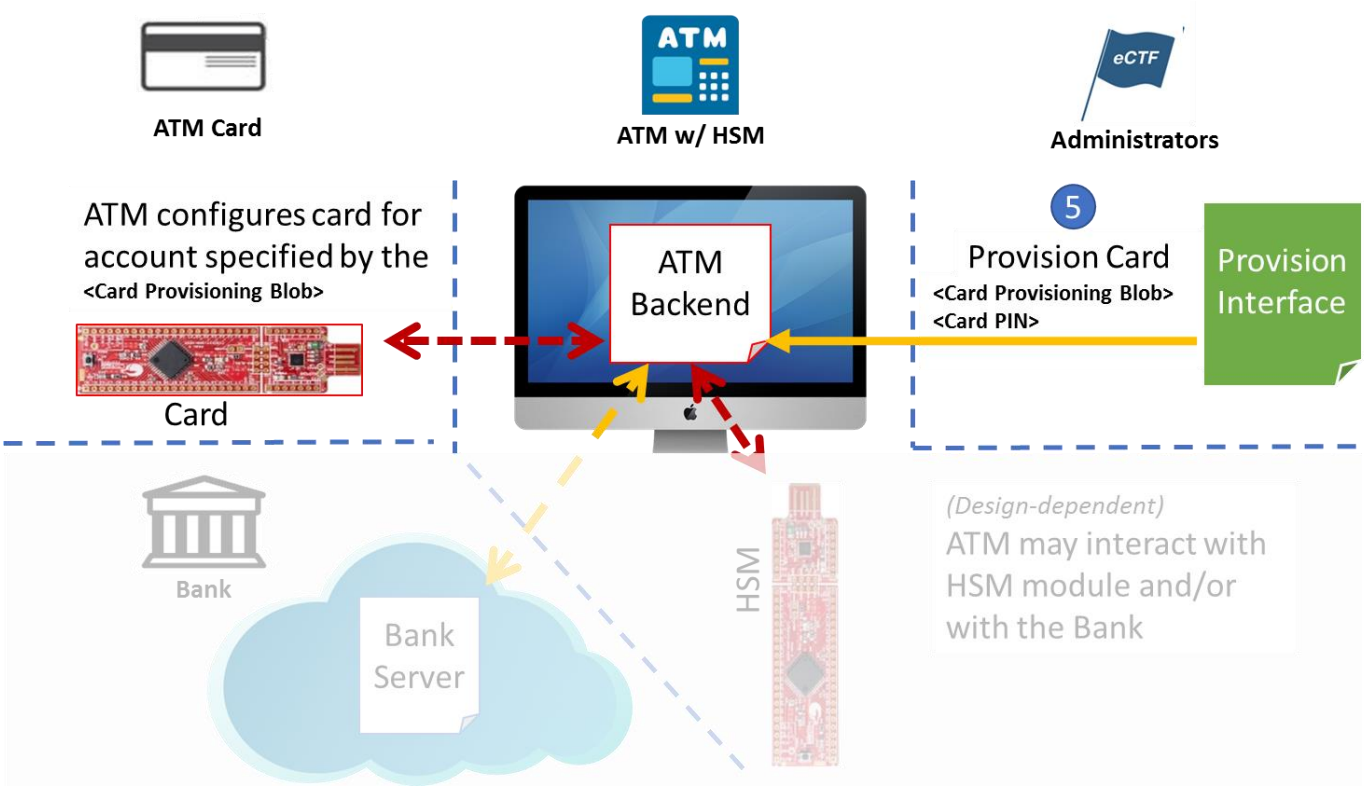

Figure 11: Admins provision new ATM Card

a. Admins send a "provision_card" command and supply the **<Card Provisioning Blob>**.
b. (optional) ATM may interact with ATM Card, Bank Server, and HSM to perform additional initialization steps

Provisioning operations are all considered to be performed in a secure "factory" at MITRE, and aren't meant to be the focus of the competition. Any provisioning blobs generated in the factory will not be freely available to attackers. Provisioning material and protocols should be designed such that attackers are not able to create ATMs and ATM Cards that are recognized by the Bank Server (since they do not have access to the Bank Server Administrator Interface, and therefore no access to provisioning material). Additionally, it is assumed that an attacker will not have access to the ATM, HSM, and ATM Cards until after they have been programmed with hardware protections enabled. After the initial provisioning process, attackers will have physical access to an ATM HSM and multiple ATM Cards. Therefore, physical attacks on the microcontroller are fair game.

4.6 Other Considerations

A final wrinkle is that at some point a hacker could breach the cloud service hosting the Bank Server Docker container and released the container onto the Internet. **Therefore, it should be assumed that all attackers will have access to an old snapshot of the Bank Server.** The released container held many legitimate, fully-provisioned accounts that have all since been closed, but some potential information held within may still be valuable (such as stored PINs). All material provided in the Attack Phase is associated with a brand new bank server created after this data breach occurred. A cleverly designed system will minimize the damage caused by this data breach.

5 Handoff Phase

After the completion of the Secure Design Stage, each team must submit their design to MITRE. Details of the exact submission process will be provided closer to the Handoff date. A completed design must include:

- Source code, documentation, and Creator project for both the HSM and ATM Card
  - Both projects should be listed in the provided `ectf-workspace.cywrk` file
- Source code and documentation for the Bank Server
- A Dockerfile and Makefile to build and run the Bank Server
- A Dockerfile and Makefile to build and run the ATM Backend Server

These files must be presented in the following directory structure:

- `ectf-workspace.cywrk`
- `CARD.cydsn/`
  - `CARD.cyprj`
- `SECURITY_MODULE.cydsn/`
  - `SECURITY_MODULE.cyprj`
- `bank_server/`
  - `Makefile`
    `Dockerfile`
- `atm_backend/`
  - `Makefile`
  - `Dockerfile`

After receiving a team's design, MITRE will build each tool (if necessary) and validate that all system components meet the functional requirements. Any design that will not build or does not meet these requirements will not be able to progress to the attack stage of the competition. MITRE will contact each team with handoff status as quickly as possible (typically within two business days) after a team's submission, whether it is accepted as functional or not. If a team's project 'Makefile's have the 'logs' target operation, competition organizers will return debug logs to the team upon failure. Teams can resubmit updated designs after previous designs were not accepted. Once a team's design has been accepted, further updates will not be accepted.

All source code and documentation, as well as the build environment/Dockerfiles, will be provided to other teams during the attack stage to discourage security-by-obscurity, as well as to accelerate attack development and encourage more sophisticated techniques for both sides. Additionally, particularly good documentation will be worth extra points at the discretion of the MITRE competition committee.

All teams that have had their designs accepted during the Handoff Stage may progress to the Attack Stage.

6 Attack Phase

Each design that has been validated during the Handoff Stage is available for attack. For each design, the files listed above in *HandOff Phase* will be made available to all attacking teams. Additionally, teams will be provided with a set of dummy provisioning material and flags. Finally, teams will receive a set of recordings from a card "skimming" attack. Teams should use these files to develop proof-of-concept attacks using your own development hardware.

Once a team has developed an attack against a specific design (that they believe will capture a flag) they may request a system provisioned with that design from MITRE. This provisioned system will contain the real flags to be protected by the designing team. At any point in time a team may only have two provisioned systems that they have yet to capture a flag from. Once a flag has been captured, a new provisioned system from another team may be requested. Due to the limited number of un-attacked systems a team may have out at a time, systems should only be requested after a proof-of-concept attack has been developed. Further details of this request process will be provided closer to the Attack Stage.

We strongly encourage responsible disclosure[22] if any vulnerabilities are discovered on open-source or commercial components used as part of the system. The MITRE team can help to coordinate the responsible disclosure of weaknesses to appropriate parties.

When a provisioned system is requested, MITRE will provide the following artifacts:

- The IP address and port of the Bank Server
  - This server and IP address/port will be unique for the requesting team
- A provisioned ATM Docker container and associated HSM (PSoC dev board)
- Provisioned ATM Cards (PSoC dev boards)
  - **"Own Card"** (for the Known PIN flag)
  - **"Stolen Card"** (for the Unknown PIN flag)
- The full Docker container representing the "Hacked Server" (for the "Data Breach" flag)
- Recordings of transactions involving the "Skimmed Card" (for the "Skimming" flag)
  - Includes all serial and network traffic from the following legitimate operations:
    - Withdrawal
    - Balance check
  - The PIN used in the recordings for the **"Skimmed Card"**
- Instructions for how to return the "Own Card" to MITRE, allowing it to become the **"Borrowed Card"**

[22] https://en.wikipedia.org/wiki/Responsible_disclosure

6.1 Flag Descriptions

During the attack phase, target systems will be loaded with "flags", which are ASCII strings that can be submitted to the live scoreboard to score points. More details on how to obtain and submit flags will be provided upon entering the Attack phase. The following table provides a brief description of each of the competition flags.

Flag Name	Capturing this flag proves that you can…	Security Goal
Cloning *Read-Access*	…determine the account balance of the **"Borrowed Card"** account.	Two-Factor Authentication Pt. 1
Cloning *Write-Access*	…withdraw money from the **"Borrowed Card"** account.	Two-Factor Authentication Pt. 1
PIN Bypass *Read-Access*	… determine the account balance of the **"Stolen Card"** account.	Two-Factor Authentication Pt. 2
PIN Bypass *Write-Access*	…withdraw money from the **"Stolen Card"** account.	Two-Factor Authentication Pt. 2
Skimming *Read-Access*	… determine the account balance of the **"Skimmed Card"** account.	Secrecy and Authenticity
Skimming *Write-Access*	…withdraw money from the **"Skimmed Card"** account.	Secrecy and Authenticity
ATM Exploit	…withdraw more cash from the ATM than you were given or withdraw money without a card.	Software Assurance
PIN Extraction	…extract PIN from the **"Stolen Card"** (may not be possible for systems that do not store PINs on the card)	PIN Secrecy (Card)
Data Breach	…determine all PINs from the provided "Hacked Server" container (may not be possible for systems that do not store PINs in the Bank Server)	PIN Secrecy (Bank)

7 Scoring

Points are scored in one of the five ways listed below.

7.1 Design Phase Milestone Flag Points

To encourage teams to stay on schedule during the design phase and to give the organizers insight into each team's progress, a small number of flag points will be awarded for reaching certain milestones. The milestone flags are:

Milestone	Proof
Read the rules	Submit the flag "I totally read the rules. All of them. I promise." when the scoreboard opens.
Build + Program + Serial	Build and program the "Milestones Demo" project to the PSoC and read the flag from the serial interface.
Debug	Build and program the "Milestones Demo" project to the PSoC and read the in-memory flag using the debugger.
Network Plumbing (socat)	Build and program the "Milestones Demo" project to the PSoC and connect it to the server (network address of the server provided with the milestones demo code) using socat.

Please note that while it may be possible to obtain the "proof" flags by reverse engineering the "Milestones Demo", reverse engineering is not the intended method of capture. If you focus on getting your development environment up and running the milestone flags should be very little additional effort.

The "Milestones Demo" project will be made available at the first Kick-off meeting.

7.2 Offensive Flag Points

Each system is required to hold and protect "flags" that should only be revealed if the system is compromised. By submitting flags, a team is demonstrating that they have compromised the target system. A brief description is required for each attack that results in a flag submission. The point value of any given flag will be adjusted dynamically and automatically based on multiple factors:

- If multiple teams capture the same flag, then the value of that flag will be divided among all the teams that capture it (distribution is not equal – it is determined based on time of capture). Naturally, more difficult attacks will be executed by fewer teams and therefore rewarded with more points.
  Note: Your total score will drop each time another team captures a flag that you had already captured. This is because the flag points that you are initially awarded need to be re-distributed as additional teams capture the same flag.

  

  Although counter-intuitive, it may be a good strategy to seek out and spend time attacking the most difficult targets. The most challenging flags will, in theory, be worth the most points in the end.

- The number of points a flag is worth increase over time as it remains un-captured. This will make the difficult flags more and more appealing as the competition goes on.

- To prevent teams from "holding" onto a flag without submitting it, the team that captures each flag first will get more points for that flag than teams that capture it later.

7.3 Defensive Flag Points

Points will be awarded for every flag that has not been captured by other teams at a regular time interval (e.g. every day). As a result, more secure designs are likely to accrue more points than other designs. Additionally, only designs that have completed the Handoff phase are able to accrue defensive points, so teams that take longer to submit a working design may score fewer points.

7.4 Documentation Points

Good documentation will be rewarded to discourage security-by-obscurity. "Good documentation" is meant to describe clear and well-commented code, useful descriptions of modules/functions/classes, and other documents that clearly describe how to read or approach the entire code base.

We are not looking for justification of your design or lengthy documents detailing your implementation in excruciating detail. ***A concise and clear README.md, combined with well-structured and well-commented code can be sufficient for Max points. Quality will be valued over quantity.***

The maximum number of points that can be scored for documentation is equal to the value of an offensive flag scored on the last day of the competition, with the actual amount being a percentage of that maximum:

- **Max** Exemplary documentation, comments, and code structure that is clear and easy to understand.
- **75%** Good comments and high-level documentation
- **50%** Good comments, but lack of clear high-level documentation
- **25%** Confusing code and little or no actual documentation
- **0%** Very confusing or deceptive comments and documentation

Points for documentation will not be awarded until near the end of the attack phase to allow for proper analysis. Honest feedback on documentation from other teams will be solicited and will be factored into the final point determination.

7.5 Write-ups

There will be an opportunity for the top teams to provide write-ups for additional points. These teams will have an opportunity to submit a defensive write-up as well as a single attack write-up. The defensive write-up may discuss security measures that worked well, those that could have been improved upon, or any that were planned but could be developed in the time provided. The attack write-up is to award teams that develop interesting or novel attacks which do not directly capture an existing flag. Further details on the number of teams that may submit write-ups and the content/format of the write-ups will be provided during the attack phase.

8 Important Dates

Kickoff --- January 17th, 2018

- Competition officially kicks off.

System Handoff --- March 1st, 2018

- System design and implementation is due.
- After MITRE has verified a submitted design, the designing team will be given access to all other verified designs for attack.

Scoreboard Closes --- April 13th, 2018

- Flag submission is closed.
- Teams will be contacted for write-ups, which will be due April 18th

Award Ceremony – April 19th, 2018

- The top scoring teams will be invited to MITRE to present their work at an award ceremony, where MITRE will announce the results of write-up judging and present awards.

9 Rules

Most rules are described and explained throughout the challenge description in the earlier sections, but this section serves as a concise summary of the most important rules.

(1) In addition to the rules provided by MITRE, participants should also adhere to all the policies and procedures stipulated by their local organization/university and state and federal laws.

(2) MITRE reserves the right to update, modify, or clarify the rules and requirements of the competition at any time, if deemed necessary by the eCTF admins.

(3) When submitting your secure design, all source code and documentation must be shared.
   (a) This is to discourage security-by-obscurity, as well as to accelerate attack development and encourage more sophisticated techniques for both sides.

(4) During the attack phase, only attack the student-designed system components explicitly designated as targets (see Figure 5 for specific components that may be attacked).

(5) All flags must be validated by submitting a brief description of the attack.
   (a) Attack descriptions should be sufficiently detailed to allow the system designer to correct their vulnerability.
   (b) eCTF admins may invalidate points for flags that are not validated before the completion of the eCTF.

(6) No permanent lock-outs are allowed. Your system must meet some minimum performance requirements:
   (a) No ATM operation may take longer than 5 seconds.
   (b) Provisioning of the ATM/HSM or card cannot take longer than 120 seconds.
   (c) Intentional delays of longer than 100 ms are forbidden (except when the system detects it is under attack, in which case rule (6)(a) still applies).

(7) Team sizes are unlimited.
   (a) Most teams will consist of members of varying degrees of experience and skill level. Our hope is that this creates an opportunity for mentoring, where the most knowledgeable team members will help teach and guide the other members of the team. Team advisors should help manage meeting times and organization of large teams.
   (b) We want to encourage as many students to participate as possible, even if they are not willing to commit a significant amount of time to the competition.
   (c) An unlimited team size is fairer since enforcing a limited team size would be extremely difficult.

(8) Teams may consist of students at any level: undergraduate, graduate, PhD, or a mix.

(9) Teams are limited to two write-up submissions.

If the eCTF admins determine that a team has violated any of the rules, the team may be disqualified from the competition.

If you have any questions, ask!

Email us at ectf@mitre.org

10 Frequently Asked Questions

10.1 Is it OK to obfuscate our source code to make it more challenging to understand and attack?

No. Obfuscations performed at compile-time (e.g. to make binary reversing more challenging) is OK, but your source code needs to be written in a clear and maintainable fashion. It should be well commented and/or otherwise documented clearly.

10.2 Can we add intentional delays during ATM operations to make it more difficult for an attacker to collect large numbers of observations?

Under normal use, there should not be any intentional delays that are noticeable to the user because a slow ATM will negatively impact the user experience (in the real world, this could result in lost customers). For our purposes, we'll consider any intentional delays more than 100 milliseconds to be unacceptable to the user.

If your system detects that it is under attack, additional delays are OK, but must be limited to no more than 5 seconds per operation (an operation is any command initiated by the bank administration and ATM customer XML-RPC interfaces). Commands initiated by the ATM provisioning interface may need to take longer (we'll allow up to 120 seconds). Permanent lock-outs or self-destruction is not allowed (see next question).

10.3 Is it OK to brick any component of the banking system when an attack is detected?

No! All components must remain functional throughout the competition.

10.4 Can we Denial-of-Service (DoS) any of the servers?

No. Trying to take down any team's Bank Server is not allowed as part of this competition. This includes, but is not limited to flooding the network card with traffic or sending "kill" packets designed to brick the server itself. DoS attacks are also not particularly useful in the competition, because each team gets their own unique server during the Attack Phase, so taking down a server only prevents you from capturing flags. If you inadvertently take down your personal server, the eCTF organizers will attempt to restart the server at their earliest convenience.

10.5 Can we physically modify the chip with countermeasures?

No. During the Attack phase, we provision the chips that teams will attack, so you won't have an opportunity to physically modify the chip during provisioning. Everything that needs to be done to the chip during provisioning needs to be done in an automated fashion within the defined provisioning sequence outlined above. Modifying requested provisioned hardware sent by the organizers during the Attack Phase is allowed (and encouraged).

10.6 How many hardware modules can we get during the attack phase? (e.g. if we keep bricking them, can we keep getting new ones?)

Each team will be limited to having two provisioned systems at a time (which means three hardware modules per system). Once you've successfully captured at least one flag from one of the systems, you may request an additional system. In some cases, MITRE may require you to return the modules that were already attacked.

10.7 Can we attack the other teams' development environment?

No! Only the provisioned hardware, team designed protocols, and server containers are considered in-bounds. In other words, there is **nothing** that you are allowed to attack until you and your target are in the attack phase.

10.8 Is social engineering in-scope for this competition? Can we send phishing communications to other teams to trick them into revealing their secrets?

No, please don't do this. Keep your attacks technical. We love creative ideas, but this one can easily violate university, state, and federal regulations.

10.9 Can we store bills in the Bank server and only send them to the ATM after the ATM user has properly authenticated?

No. The strings used as "bills" are intended to represent physical dollar bills and therefore must be stored in the ATM or HSM.

10.10 Can we encrypt the bills so that they are only stored in the ATM as ciphertext? We would transmit the key for decrypting them from the Bank server after the ATM user has properly authenticated.

No. Storing the bills as ciphertext and keeping the key only on the Bank Server is essentially the same thing as storing part of the bills in the Bank server.

10.11 Is the order that bills are dispensed important?

Yes! To capture the "ATM Exploit" flag, an attacker must withdraw more bills than would normally be possible with the account associated with the "Own Card". This will only be difficult if bills are dispensed in precisely the order they are provisioned.

10.12 How does the "Own Card" become the "Borrowed Card"?

During the attack phase, you may request that your "Own Card" become the "Borrowed Card". The organizers will provide you with information on how to return the PSOC to MITRE. Upon receipt of the card, the organizers will deposit additional funds into the account associated with the card, at which point it will be considered the "Borrowed Card".

10.13 Will the PIN change when the "Own Card" becomes the "Borrowed Card"?

No, the PIN does not change when the "Own Card" becomes the "Borrowed Card". Note that the "Borrowed Card" differs from the "Stolen Card" in that the attackers know the PIN associated with the account.

11 Revisions

1.0 – Initial Release

1.1 – Updates for additional interface requirements and additional details for clarity on rules and requirements

- Modified sections 4.2.2 and 4.3.2 for additional interface functions that must be supported to simplify testing.
- Modified sections 4.3 and 4.3.3 for additional clarity on bill storage.
- Added questions and answers to FAQ (10.9, 10.10, 10.11) for additional clarity on bill storage.
- Added questions and answers to FAQ (10.12, 10.13) for additional clarity on the Borrowed Card.
- Updated answer to FAQ 10.2 for clarifying that we're OK if provisioning operations take longer than 5 seconds when needed… the intent is to disallow significant intentional delays on user interfaces – the provisioning operations should not need to be optimized for performance.
- Updated Section 9 rule (6) to clarify rules on maximum operation times and intentional delays.