Applied Ca$h Eviction through ATM Exploitation

Trey Keown
Red Balloon Security
New York, USA
trey@redballoonsecurity.com

Brenda So
Red Balloon Security
New York, USA
brenda@redballoonsecurity.com

***Abstract*—Automated Teller Machines (ATMs) are among the oldest devices to be connected to a network. Despite this, the high barrier to entry for legal reverse engineering efforts has resulted in largescale ATM deployment without the testing that would be expected of machines where the cost of compromise can be measured in the number of bills dispensed. Our research examines retail ATMs from a reverse engineer's perspective and details two network-accessible vulnerabilities we discovered as a result – a buffer overflow in the Remote Management System (RMS), and a remote command injection via the eXtensions for Financial Services (XFS) interface. These vulnerabilities can lead to arbitrary code execution and jackpotting, respectively.**

I. Introduction

While the ATMs owned and operated by banks, financial ATMs, often have a more compelling case to be well-secured, the retail ATMs found scattered across gas stations and convenience stores are often an offering more focused on getting the price right. One such cost-effective ATM is the Nautilus Hyosung HALO II [1].

Our initial interest in the ATM came by the virtue that, as a computer which dispenses money, it is an attractive platform to use as a base for an information security challenge [2]. Initial work involved creating a payment processor for handling ATM transactions – for this, we developed a server supporting the Triton Standard [3]. Further work involved reverse engineering the ATM itself. This was aided by the availability of a firmware update, a JTAG port that was accessible with hardware modifications, and a lack of signature checking prior to applying full firmware updates.

II. Initial Reverse Engineering

Our target ATM is based on an architecture reused across a number of the Nautilus Hyosung's other price-conscious ATMs – Windows CE 6.0 (officially end-of-life as of April 10, 2018) running on an 800MHz ARM Cortex-A8. The platform of libraries and applications running on top of this is referred to as MoniPlus CE.

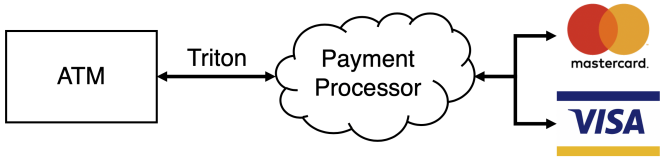

Figure 1: An example of how a transaction from an ATM is passed along to card networks.

In order for the ATM to function normally, it requires a connection to a payment processor. For retail ATMs, a payment processor is typically a third-party vendor that handles upstream interactions with banks, as shown in Figure 1. This communication can be achieved via either TCP or dial-up. A number of protocols can be used over this link, but the one we found most readily accessible was the Triton Standard. A draft copy of this standard [3] is available online. While current implementations do not exactly match this draft, it nonetheless provided an initial guide to implementing a server capable of handling transactions with an ATM.

While not functional out of the box, there was an unpopulated port on the main board which resembled what would be expected of a JTAG connector. Physically nearby are unpopulated pads with markings implying a resistor should be present. After populating these with an arbitrary low resistor value and performing pin mapping with a JTAGulator [4], we obtained a fully functional debug interface.

Firmware for the ATM is publicly available online [5]. It contains the bootloader binary, the kernel binary, and a zip file containing all the application software and libraries.

III. Remote Management System (RMS)

In the context of an ATM, a Remote Management System (RMS) is a network-based administration interface for owners and administrators to manage a network of ATMs. In the ATM examined in this paper, capabilities of the RMS include (among many others) dumping an ATM's version and configuration information, collecting transaction history, and remotely updating an ATM's firmware.

Normally, an ATM administrator needs to use a client called MoniView [6] to send commands to the RMS server running on the ATM. In order to authenticate these commands, the ATM's serial number and RMS password are passed along in the RMS command packet. However, an unauthenticated attacker can send a maliciously crafted packet to the remote RMS endpoint over a network to cause a buffer overflow and corrupt structures used in the cleanup of the RMS control library, `RMSCtrl.dll`. This corruption can lead to arbitrary code execution and persistence in the ATM, which is further described in the following sections.

A. Protocol Description

RMS communication between the client and the ATM is obfuscated with values from a lookup table that are XORed

with the message plaintext. The format of the RMS packet is shown in Table 1. The encoded data contains the RMS request type as well as the ATM serial number and password that is used to verify the RMS packet.

Length	Content	Description
1 byte	STX (0x02)	RMS start byte
2 bytes	XX XX	Data length (n)
1 byte	XX	Encryption seed
n bytes	XX...	Encoded data
1 byte	ETX (0x03)	RMS end byte
1 byte	XX	Longitudinal Redundancy Check (LRC)

Table 1: Request RMS packet structure [7]

B. RMS Buffer Overflow

The vulnerability here is a buffer overflow in an function called by `CRmsCtrl::RMS_Proc_Tcp()`. This overflow is ultimately caused by a call to `memcpy` without proper bounds checking, allowing the overflow of a static buffer in the RMS control library, `RMSCtrl.dll`. Any packet sent to the RMS server of the ATM will initiate the function call in Listing 1, which in turn does the following:

```
char* recv_buffer;

CRmsCtrl::RMS_Proc_Tcp(){
  int* num_recv_char
  bool is_connect, is_recv, is_verified;
  // connects to the RMS server
  is_connect = CDevCmn::fnNET_RMSConnectAccept()
  if (is_connect){
    memset(recv_buffer, 0, 0x2800);
    // receives RMS packet
    is_recv = CRmsCtrl::RMS_Recv(recv_buffer, num_recv_char
     , 0)
    if (is_recv){
      // verifies RMS packet
      is_verified = CRmsCtrl::RMS_VerifyMsg(recv_buffer, *
     num_recv_char);
      if (is_verified){
        // parses message
      }
    }
  }
}
```

Listing 1: Pseudo-code of RMS Process

1) `fnNET_RMSConnectAccept` sets up TCP connection between the ATM and the RMS client, by default on port 5555.
2) `RMS_Recv` calls `fnNET_RMSRecvData`, which copies data from the received RMS packet over to a global receive buffer. If the packet is formatted correctly, it then proceeds to decrypt the XOR-encoded data.
3) `RMS_VerifyMsg` verifies the ATM serial number and RMS password in the decrypted data.
4) If the message is verified, the function then proceeds to parse the packet and generate a response to the RMS client.

The `fnNET_RMSRecvData` function in step 2 does not have bounds or credentials checks on the data received. Moreover, packet verification occurs after the packet has been copied over to the `recv_buffer` memory location. Thus, as long as the packet adheres to the structure in Table 1, its data is copied over indiscriminately. Any packet larger than 0x2800 bytes will result in a buffer overflow.

C. Arbitrary Code Execution

The aforementioned overflow eventually overwrites a function pointer that gets invoked when the ATM is shutting down. We also found that the `.data` section of `RMSCtrl.dll` is executable, thus we are able to write shellcode that gets executed when the DLL exits. Because there are regions of memory after the overflowed buffer that are never overwritten and aren't critical for system operation, this shellcode can remain in memory until the device powers off. When the main ATM application exits cleanly, such as when a technician performs a firmware update on the ATM, the shellcode is executed.

D. Persistent Memory Modification

Through arbitrary code execution, we achieved persistence by modifying memory on the ATM's Nonvolatile Random-Access Memory (NVRAM) chip. NVRAM is used to store network and configuration information of the ATM, such as enabling or disabling SSL, specifying the payment processor's IP address, and storing passwords. The NVRAM can be accessed through two pairs of API functions – `MemGetStr` and `MemGetInt` retrieve information from NVRAM, while `MemSetStr` and `MemSetInt` update information on NVRAM. The aforementioned shellcode can achieve goals useful to an attacker by updating the ATM's configuration to disable SSL, redirecting transactions to a malicious payment processor, or changing passwords to a value known by the attacker. Since this configuration data persists across reboots, modifications to the NVRAM provide a simple way for an attacker to apply persistent, malicious changes.

IV. CEN eXTensions for Financial Services (XFS)

The European Committee for Standardization (CEN) is the maintainer of a standard for ATMs called eXtensions for Financial Services, or XFS. This standard is derived from one originally created by Microsoft in an effort to create a common platform for financial devices running on Windows, a role the standard still fulfills to this day. XFS plays an important role in the ATM industry, acting as the platform targeted by both high-level and low-level software running on ATMs. It exposes a homogenized interface for working with different components in ATMs (and other financial devices), such as pinpads, cash dispensers, and card readers.

A. Introduction to XFS

XFS defines a set of interfaces with the aim of unifying the ways in which financial applications on Microsoft Windows interact with related pieces of hardware. This is accomplished through a client/server architecture as seen in Figure 2. The financial frontend interacts with the XFS API, while the

service providers, which handle interactions with hardware, use the corresponding XFS Service Provider Interface (SPI). The translation and dispatch of these messages is handled by the XFS Manager. For our purposes, we will refer to all the vendor-created components of XFS as the XFS middleware. The XFS middleware implementation used with MoniPlus CE is called Nextware [8].

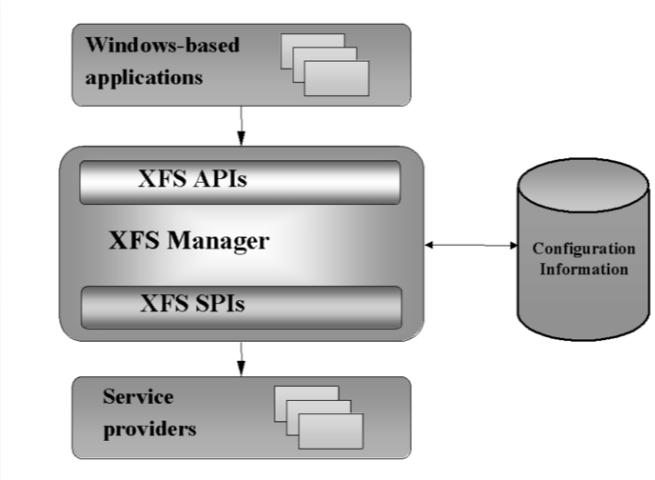

Figure 2: XFS client/server architecture as seen in the reference documentation [9]

There are a number of device classes defined as a part of the standard. A listing with corresponding abbreviations is shown in Table 2.

Service Class	Class Name	Class Identifier
Printers	PTR	1
Identification Card Units	IDC	2
Cash Dispensers	CDM	3
PIN Pads	PIN	4
Check Readers and Scanners	CHK	5
Depository Units	DEP	6
Text Terminal Units	TTU	7
Sensors and Indicators Units	SIU	8
Vendor Dependent Mode	VDM	9
Cameras	CAM	10
Alarms	ALM	11
Card Embossing Units	CEU	12
Cash-In Modules	CIM	13
Card Dispensers	CRD	14
Barcode Readers	BCR	15
Item Processing Modules	IPM	16

Table 2: XFS Device Classes

A number of actions associated with each device class are defined as part of the XFS standard. For example, the devices defined by the standard which are responsible for working with credit and debit cards are called Identification Card units (IDCs). The IDC standard defines constants for commands that can be executed such as `READ_TRACK`, `EJECT_CARD`, and `CHIP_IO` [10]. These commands will be called when the primary application (in this case, `WinAtm.exe`) needs to control or query a device.

Due to the fact that XFS exposes a homogenized interface for interfacing with different financial devices, it becomes not only an attractive target for the ATM industry, but also for malware authors. Numerous ATM-related pieces of malware, such as GreenDispenser [11] and RIPPER [12], utilize XFS for interfacing with the card reader, pinpad, and cash dispenser. These interactions are performed in a payload after being dropped on the ATM by some other means, normally by taking advantage of physical access to the device.

B. Nextware – XFS Middleware Implementation

In order to facilitate XFS message passing between different components that make up the XFS middleware, Nextware uses TCP socket-based Inter-Process Communication (IPC). This is a reasonable choice given the lack of many standard Windows IPC features on Windows CE [13]. However, a problem arises when these sockets are misconfigured. When creating the sockets, the listening address is given as `0.0.0.0` instead of `127.0.0.1`, meaning that instead of listening only to IPC messages on the loopback device, the local server will listen to messages from any network device. This misconfiguration is reflected in a port scan, shown in Table 3.

Port	Open By Default	Description
80	Yes	Default Windows CE Webserver
443	Yes	Default Windows CE Webserver
5555	No	Remote Management Service (RMS)
8001	Yes	SIU (Sensors, Indicators, etc.)
8003	Yes	IDC (Card Reader)
8004	Yes	CDM (Cash Dispenser)
8006	Yes	PIN (PIN Pad)
8010	Yes	PTR (Receipt Printer)

Table 3: Open TCP Ports

While the purpose of these ports was not initially understood, a scan of the registry provided a strong hint. A number of XFS configuration keys are stored in the `HKEY_USERS` registry hive. An excerpt of the configuration for the cash dispenser device class is shown in Listing 2.

```
CashDispenser\Provider:CashDispenser (String)
CashDispenser\port:8004 (DWord)
CashDispenser\Class:CDM (String)
CashDispenser\Type:CDM (String)
```

Listing 2: Excerpt of registry entries for one peripheral under `HKEY_USERS\.DEFAULT\XFS\LOGICAL_SERVICES`

Attempts to obtain output from these ports was initially unsuccessful - any unknown message causes the port to refuse further input. Further analysis revealed that messages outside the expected format caused the IPC mechanism to be put in an unknown state. Complicating efforts to integrate with the sockets in this manner was a lack of any documentation on the message format used here. Capturing network traffic on the loopback interface would help with deciphering the message format required to successfully send an XFS message, but this sort of packet capture was not straightforward.

C. Dumping Network Traffic – The Hard Way

Working with Windows CE 6.0 is a complicated affair. As of 2019, the initial release of this version was 13 years ago, and the latest major release was 10 years ago. As such, any tools required becomes more difficult to source. Interfacing with the ATM in general was frustrating, though a positive note for security, due to the fact that this device ships without any keyboard or mouse drivers. Capturing network traffic on the device, while possible [14], is hindered by these complications and the fact that the image shipped on the device isn't built with this support. As such, the most straightforward method for proceeding is to identify locations where the IPC mechanism makes calls to the `socket`, `recv`, and `send` functions of Winsock 2 [15]. Since Address Space Layout Randomization (ASLR) is not active on this platform, knowledge of the loading address of these functions can be utilized across device reboots.

Tracing calls to the `socket` function exposes which underlying service owns a given socket handle used in the `recv` and `send` functions, revealing what messages are being sent and received by each service. Further analysis revealed that each message contains command data in the format defined by the XFS standard, wrapped by a header. The partially-deciphered format is shown in Table 4.

Length	Description
1 byte	XFS command type (GetInfo, Execute, etc.)
1 byte	Unknown1
2 bytes	Unknown2
4 bytes	Zero (`0x00000000`)
2 bytes	Service handle
2 bytes	Unknown3
4 bytes	Window handle
4 bytes	Unique request ID
4 bytes	Timeout
4 bytes	Timestamp
4 bytes	XFS command
Command-dependent	XFS command data

Table 4: XFS IPC Packet Structure

As an example, consider the action of setting the cash dispenser lights to flash quickly. Using the SIU specification [16] as a guide, the XFS command field would be set to `WFS_CMD_SIU_SET_GUIDLIGHT`, and the command-dependent data would be populated by filling in the corresponding structure, `WFSSIUSETGUIDLIGHT`. The values (`WFS_SIU_NOTESDISPENSER`) and (`WFS_SIU_QUICK_FLASH`) would be used to indicate that the cash dispenser should now be set to flash quickly. The values and structures referenced are reproduced in Listing 3.

Using the knowledge of the packet structure, and with a dump of all messages sent over the IPC sockets obtained via JTAG, it is possible to find messages with commands of interest (for example, cash dispensing via `WFS_CMD_CDM_DISPENSE`), fix up the timeout and timestamp fields of the packet, and replay it to trigger the action.

```
#define WFS_SERVICE_CLASS_SIU (8)
#define SIU_SERVICE_OFFSET (WFS_SERVICE_CLASS_SIU * 100)
#define WFS_CMD_SIU_SET_GUIDLIGHT (SIU_SERVICE_OFFSET + 6)

typedef struct _wfs_siu_set_guidlight
{
    WORD            wGuidLight;
    WORD            fwCommand;
} WFSSIUSETGUIDLIGHT, *LPWFSSIUSETGUIDLIGHT;

#define WFS_SIU_NOTESDISPENSER (2)
#define WFS_SIU_QUICK_FLASH (0x0010)
```

Listing 3: Referenced SIU definitions as they appear in the latest CEN/XFS standard [16]

D. XFS Attack Implications

This attack enables the ability to perform command injection over the XFS message sockets, which are visible to any device on the same local network. While this does not necessarily lead to arbitrary code execution, it does expose a large surface area over a network in the form of the unauthenticated XFS API, which has commands that are of immediate interest to an attacker (such as dispensing cash).

V. Conclusions

Despite the tough physical exterior, the ATM referenced in this paper readily gave way to two network-accessible attacks: one pre-authentication buffer overflow allowing for arbitrary code execution (and persistence) with user interaction, and one unauthenticated XFS command injection. Although only one commonly found ATM is the subject of this paper, the problems found here are likely not unique. A high monetary barrier to entry to perform legal penetration testing on these devices remains one of their most compelling defenses.

Acknowledgment

We would like to thank Red Balloon Security for providing us the resources to investigate and reverse engineer the ATM. We would also like to thank Nautilus Hyosung for being proactive in the vulnerability disclosure process and responsive in developing a fix.

References

[1] HALO II - Hyosung America. Hyosung America. [Online]. Available: https://hyosungamericas.com/atms/halo-ii/

[2] Happy save banking corporation and laundry service. Red Balloon Security. [Online]. Available: http://happysavebankingcorporation.com/index.html

[3] "Triton terminal and communication protocol," Triton. [Online]. Available: https://www.completeatmservices.com.au/assets/files/triton-comms-msg%20format-pec_5.22.pdf

[4] Joe Grand. Jtagulator — grand idea studio. Grand Idea Studio. [Online]. Available: http://www.grandideastudio.com/jtagulator/

[5] Software updates — atm parts pro. ATM Parts Pro. [Online]. Available: https://www.atmpartspro.com/software

[6] Terminal management - hyosung america. Hyosung America. [Online]. Available: https://hyosungamericas.com/softwares/terminal-management/

[7] Barnaby Jack, "IOActive Security Advisory - Authentication Bypass In Tranax Remote Management Software." [Online]. Available: https://ioactive.com/wp-content/uploads/2018/05/Tranax_Mgmt_Software_Authentication_Bypass.pdf

[8] Cen/xfs. Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/CEN/XFS#XFS_middleware

[9] Extensions for financial services (xfs) interface specification release 3.30 - part 1: Application programming interface (api) -service provider interface (spi) - programmer's reference. European Committee for Standardization. [Online]. Available: ftp://ftp.cen.eu/CWA/CEN/WS-XFS/CWA16926/CWA%2016926-1.pdf

[10] Extensions for financial services (xfs) interface specification release 3.30 - part 10: Sensors and indicators unit device class interface - programmer's reference. European Committee for Standardization. [Online]. Available: ftp://ftp.cenorm.be/CWA/CEN/WS-XFS/CWA16926/CWA%2016926-4.pdf

[11] Meet greendispenser: A new breed of atm malware. Proofpoint. [Online]. Available: https://www.proofpoint.com/us/threat-insight/post/Meet-GreenDispenser

[12] Ripper atm malware and the 12 million baht jackpot. FireEye. [Online]. Available: https://www.fireeye.com/blog/threat-research/2016/08/ripper_atm_malwarea.html

[13] A study on ipc options on wince and windows. Few of my technology ideas. [Online]. Available: https://blogs.technet.microsoft.com/vanih/2006/05/01/a-study-on-ipc-options-on-wince-and-windows/

[14] How to capture network traffic on windows embedded ce 6.0. Windows Developer 101. [Online]. Available: https://blogs.msdn.microsoft.com/dswl/2010/03/02/how-to-capture-network-traffic-on-windows-embedded-ce-6-0/

[15] Winsock functions. Windows Dev Center. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/winsock/winsock-functions

[16] Extensions for financial services (xfs) interface specification release 3.30 - part 10: Sensors and indicators unit device class interface - programmer's reference. European Committee for Standardization. [Online]. Available: ftp://ftp.cenorm.be/CWA/CEN/WS-XFS/CWA16926/CWA%2016926-10.pdf