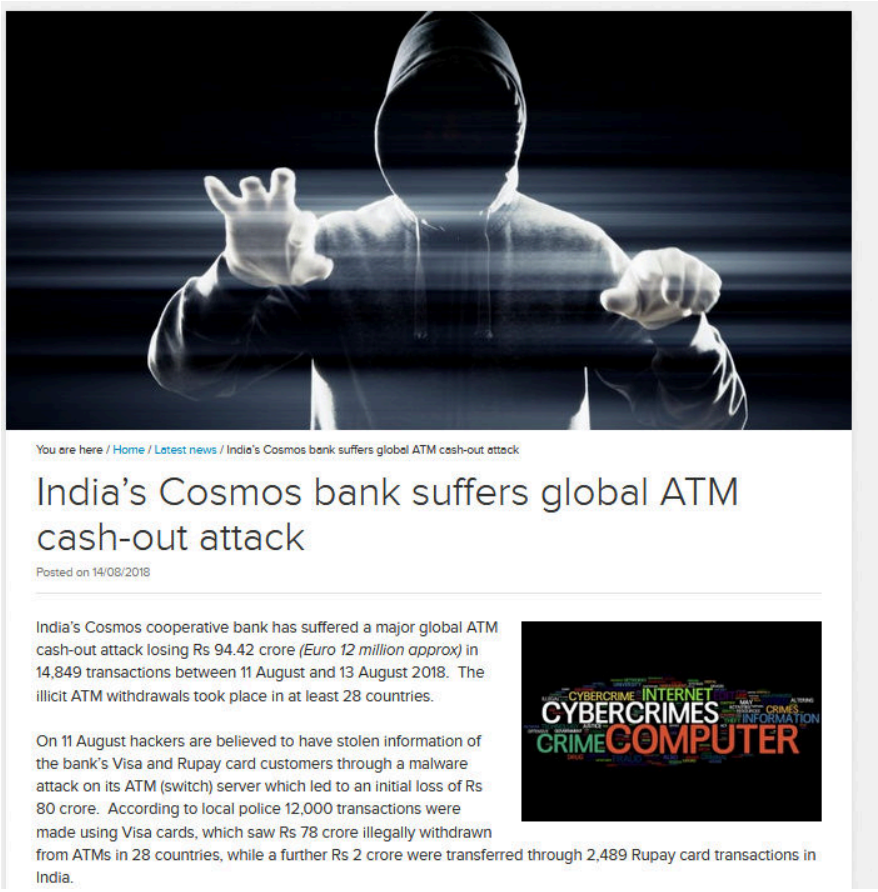
OPERATION FAST CASH
HIDDEN COBRA

Hidden Cobra's AIX PowerPC malware dissected

Frank Boldewin

Introduction

- In a highly targeted operation attackers were able to perform fraudulent transactions by compromising a bank's payment switch application server in 08/2018.
- By faking transaction response messages at the switch, actors withdrawn cash simultaneously from ATMs located in 23 countries.
- So called Fast Cash operations have been carried out by different threat groups since late 2016.
- In most of these cases attackers removed the cash limit from certain cards in the card processing system by abusing stolen bank employee credentials.
- The Cosmos Bank attack is kinda different, as actors injected malicious code into the payment switch application server running on a IBM AIX PowerPC machine.
- From a reverse engineer’s perspective it’s quite interesting to analyze such rare pieces of malcode. ;-)



You are here / Home / Latest news / India's Cosmos bank suffers global ATM cash-out attack

India’s Cosmos bank suffers global ATM cash-out attack

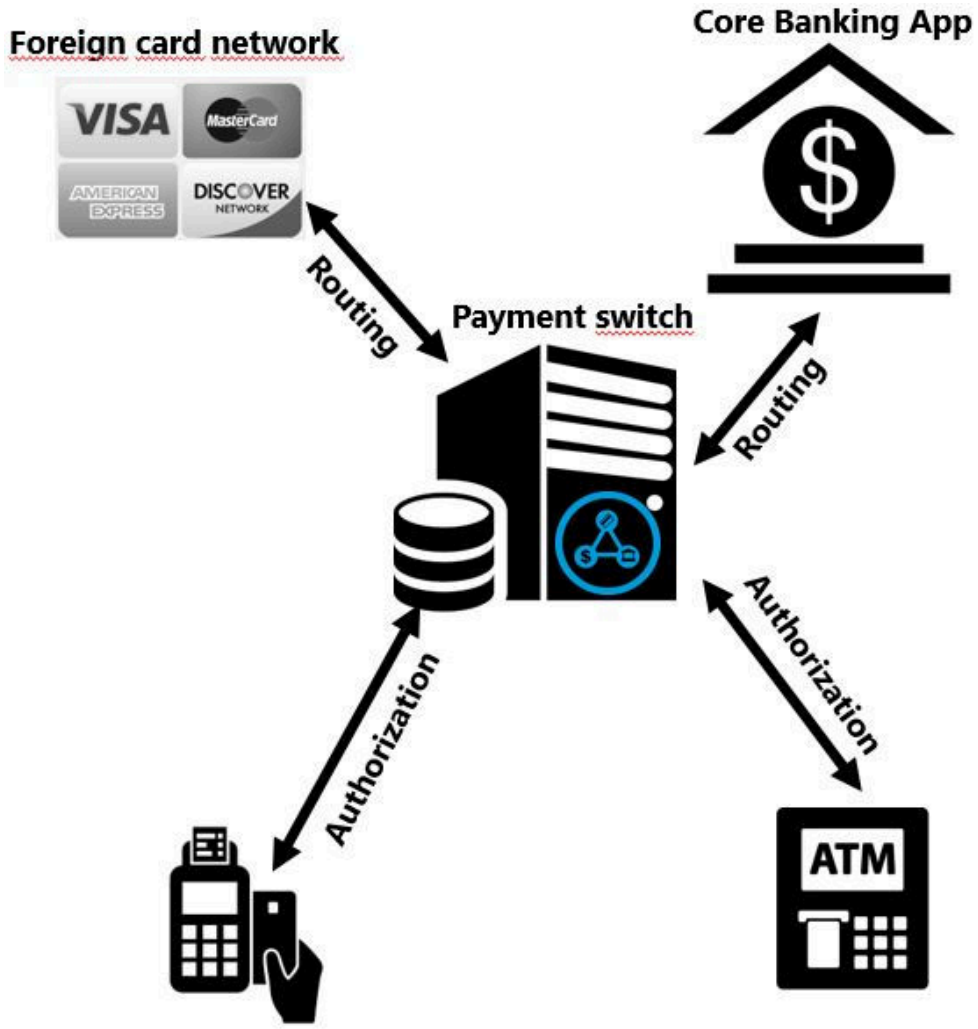
Posted on 14/08/2018

India’s Cosmos cooperative bank has suffered a major global ATM cash-out attack losing Rs 94.42 crore *(Euro 12 million approx)* in 14,849 transactions between 11 August and 13 August 2018. The illicit ATM withdrawals took place in at least 28 countries.

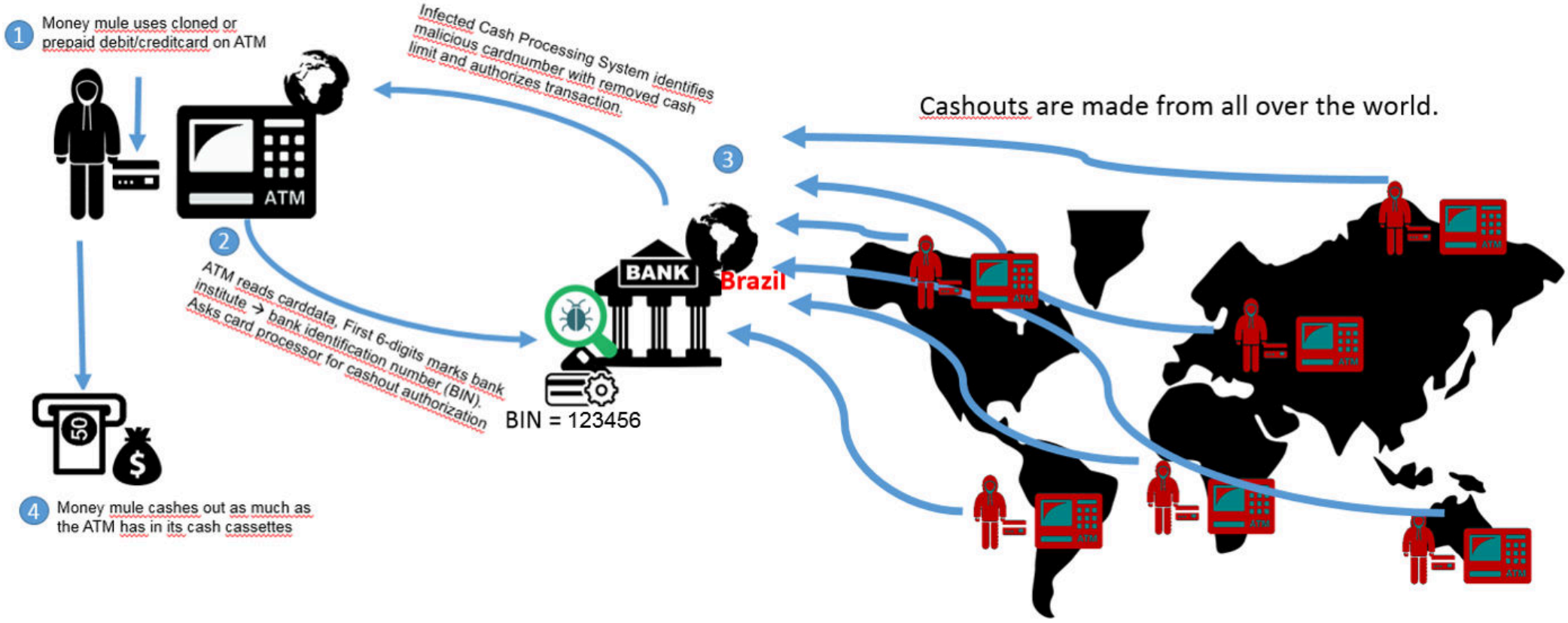
On 11 August hackers are believed to have stolen information of the bank’s Visa and Rupay card customers through a malware attack on its ATM (switch) server which led to an initial loss of Rs 80 crore. According to local police 12,000 transactions were made using Visa cards, which saw Rs 78 crore illegally withdrawn from ATMs in 28 countries, while a further Rs 2 crore were transferred through 2,489 Rupay card transactions in India.

Payment switch basics

- Primary purpose is to perform transaction processing and routing decisions
- On-us to primary authorizers
- Switching foreign transactions to EFT (electronic funds transfer) networks like Visa, Mastercard etc.
- PIN validation



Attacks on card processing illustrated



Attribution

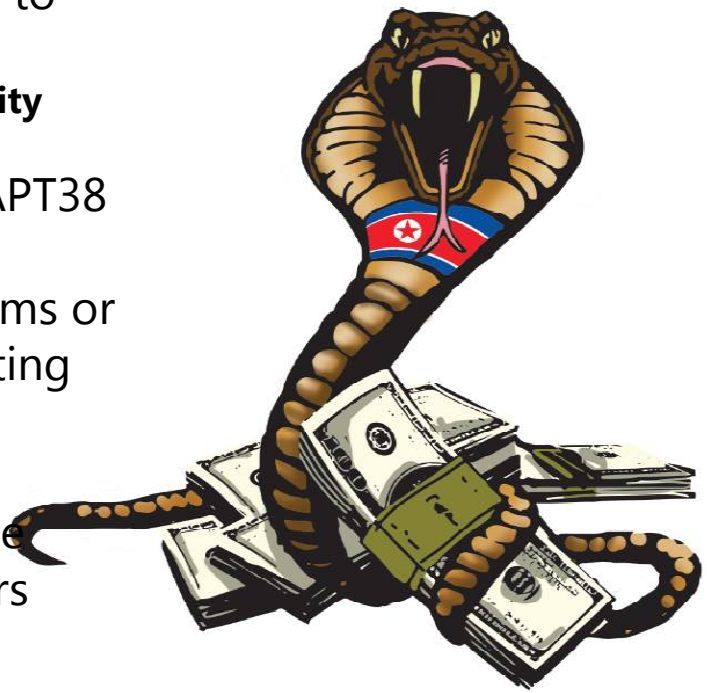
So who is suspected behind the attack?

According to the U.S. Government the malicious cyber activity refers to the North Korean government, known as HIDDEN COBRA.
https://www.us-cert.gov/HIDDEN-COBRA-North-Korean-Malicious-Cyber-Activity

Besides Hidden Cobra this actor group is also known as Lazarus or APT38

Several cybersecurity companies like CrowdStrike, FireEye, BAE Systems or Group-IB attributed attacks to this actor, being active globally, targeting all kind of industries (Banks, Governments, Media) since 2014.

The group has become particularly known since its cyber heist on the Bangladesh Central Bank in February 2016, in which 81 million dollars were captured, by attacking the SWIFT systems of the bank.

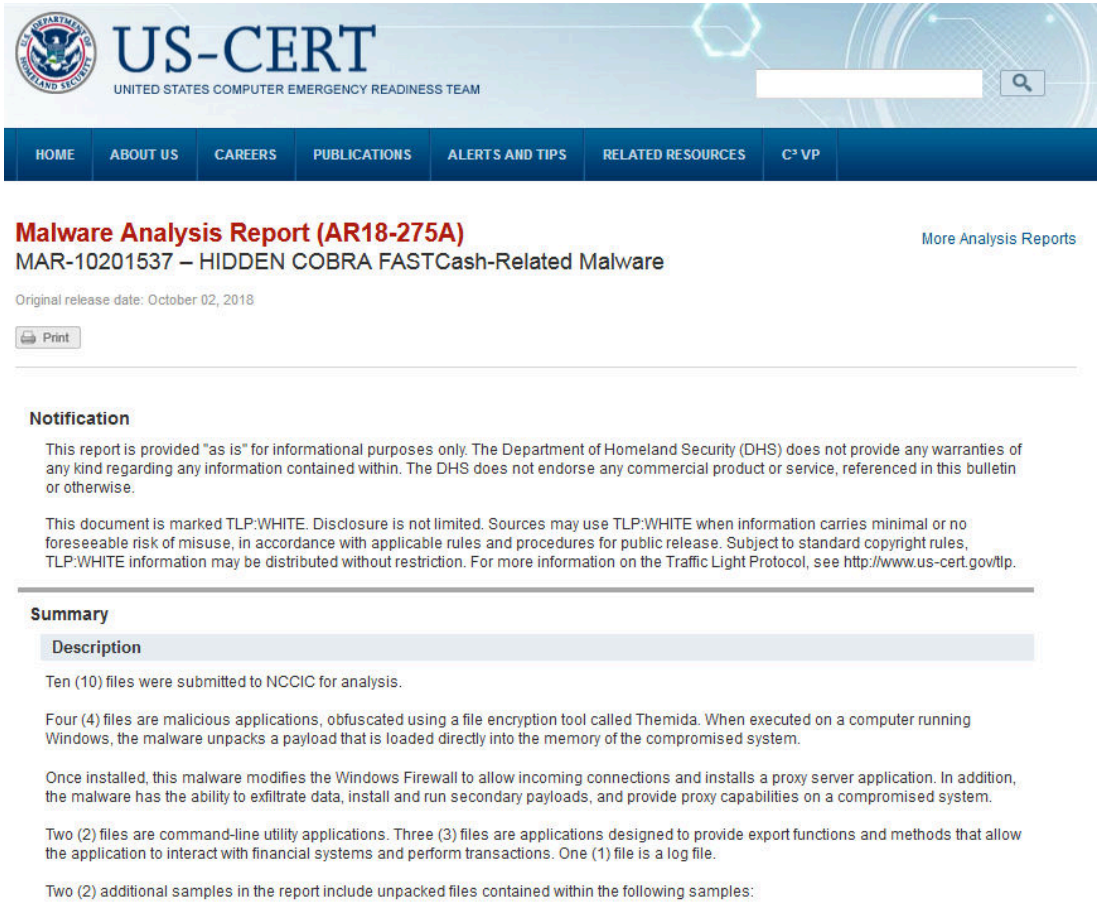


AIX (XCOFF file format) hashes from US-CERT Malware Analysis Report

- US-CERT MAR AR18-275A lists 4 samples of interest
  - Three shared object (.so) libraries used to manipulate transaction messages
  - One executable to inject .so files into target application (payment switching software)
  - All samples available on Virustotal

Injection tool
d465637518024262c063f4a82d799a4e40ff3381014972f24ea18bc23c3b27ee

Libraries
ca9ab48d293cc84092e8db8f0ca99cb155b30c61d32a1da7cd3687de454fe86c
10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba
3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c



US-CERT
UNITED STATES COMPUTER EMERGENCY READINESS TEAM

HOME | ABOUT US | CAREERS | PUBLICATIONS | ALERTS AND TIPS | RELATED RESOURCES | C³ VP

Malware Analysis Report (AR18-275A)
More Analysis Reports
MAR-10201537 – HIDDEN COBRA FASTCash-Related Malware
Original release date: October 02, 2018
Print

Notification

This report is provided "as is" for informational purposes only. The Department of Homeland Security (DHS) does not provide any warranties of any kind regarding any information contained within. The DHS does not endorse any commercial product or service, referenced in this bulletin or otherwise.

This document is marked TLP:WHITE. Disclosure is not limited. Sources may use TLP:WHITE when information carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release. Subject to standard copyright rules, TLP:WHITE information may be distributed without restriction. For more information on the Traffic Light Protocol, see http://www.us-cert.gov/tlp.

Summary

Description

Ten (10) files were submitted to NCCIC for analysis.

Four (4) files are malicious applications, obfuscated using a file encryption tool called Themida. When executed on a computer running Windows, the malware unpacks a payload that is loaded directly into the memory of the compromised system.

Once installed, this malware modifies the Windows Firewall to allow incoming connections and installs a proxy server application. In addition, the malware has the ability to exfiltrate data, install and run secondary payloads, and provide proxy capabilities on a compromised system.

Two (2) files are command-line utility applications. Three (3) files are applications designed to provide export functions and methods that allow the application to interact with financial systems and perform transactions. One (1) file is a log file.

Two (2) additional samples in the report include unpacked files contained within the following samples:

Source → https://www.us-cert.gov/ncas/alerts/TA18-275A

IBM AIX tool PVPA

Hash → f3e521996c85c0cdb2bfb3a0fd91eb03e25ba6feef2ba3a1da844f1b17278dd2

- MAR AR18-275A lists another XCOFF64 executable called PVPA.
- This is in no way related to Hidden Cobra's Fast Cash operations, as previously assumed by researchers.
- It's part of the IBM AIX PERFPMR analysis tool-suite, generally not opened to clients.
- PVPA is an AIX kernel structure that includes the information about logical CPUs, being collected with PERFPMR to give information if a logical CPU is active or folded. More on processor folding → **http://ibmsystemsmag.com/CMSTemplates/IBMSystemsMag/Print.aspx?path=/aix/administrator/virtualization/virtual_processor_folding**
- The tool determines the address of the pvpa kernel structure to update one of its variables with the <new_value>. This enables or disables additional trace information for trace hook 419 on POWER7 based systems.

Description

This file is an AIX executable, intended for a proprietary UNIX operating system developed by IBM. Figure 8 displays strings of interest. The strings contained within the file indicate it is a command-line utility. The file is designed to update a proprietary data structure on a UNIX system known as "PVPA." The code structure in Figure 9, extracted from this application, attempts to perform a raw read of this data structure from memory.

Screenshots

.data:00000...	00000009	C	/dev/mem
.data:00000...	00000009	C	set_posn
.data:00000...	00000009	C	get_pvpa
.data:00000...	0000000A	C	init_pvpa
.data:00000...	0000000B	C	high_cpuid
.data:00000...	0000000C	C	usage_error
.data:00000...	0000000E	C	getdtablesize
.data:00000...	0000000F	C	high_cpuid=%d\n
.data:00000...	00000010	C	open kernel mem
.data:00000...	0000001C	C	cpu %d, old value = 0x%02x\n
.data:00000...	00000023	C	lseek rc=%ld, value=%ld, errno=%d\n
.data:00000...	00000027	C	Usage: pvpa [<new_value> <old_value>]\n
.data:00000...	00000028	C	lseek rc=%d, cpu=%d, posn=%d, errno=%d\n
.data:00000...	0000002D	C	cpu %d, trace flag = 0x%02x idx = 0x%016llx\n
.data:00000...	00000037	C	Invalid PVPA read, magic = 0x%08x, len = %d, cpu = %d\n

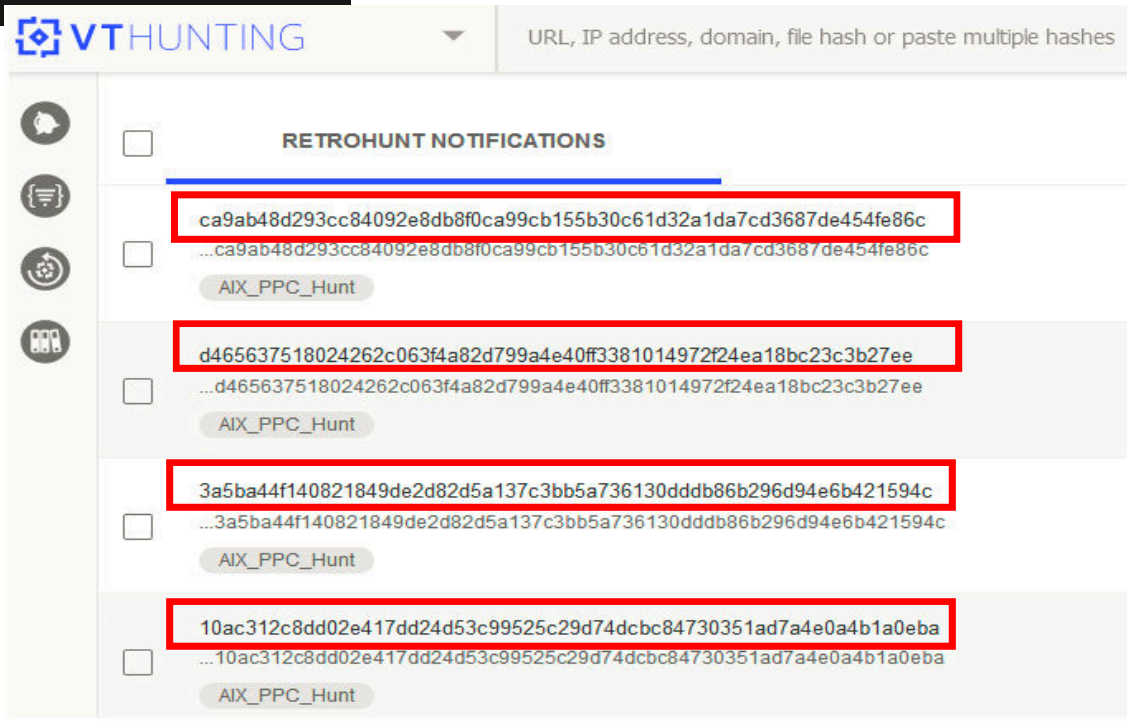
Figure 8 - Screenshot of the strings of interest

Hunting for more XCOFF files on VT (generic approach)

```
rule AIX_PPC_Hunt
{
    meta:
  description = "Generic rule to hunt for AIX PowerPC executables"
  last_modified = "2018-10-04"
  author = "Frank Boldewin"

    strings:
        $AIXPPC = "powerpc-ibm-aix" nocase ascii wide // GCC compiler artifact
    condition:
        (uint16(0) == 0xF701 or uint16(0) == 0xDF01) and ($AIXPPC) // magic bytes for XCOFF32/64 executables
}
```

Result → Only the 4 XCOFF files from the US-CERT malware analysis report are available on VT
(last try 5th Dec. 2018)

Quite rare stuff ;-)



XCOFF file format basics

- Extended common object file format (XCOFF 32/64 Bit)
  - COFF + TOC module format (Storage for addresses for global symbols)
- Provides dynamic linking and replacement of units (csects) within an object file
- Composite header consisting of:
  - File header
  - Optional auxiliary header (Information, used for loading and executing a module)
  - Section headers (Provides identification and file-accessing information)
- Raw-data sections (Information needed to dynamically load a module into memory for execution)
- Optional relocation information for individual raw-data sections
- Optional line number information for individual raw-data sections
- Optional symbol table
- Optional string table, for all symbol names in XCOFF64 and for symbol names >8 bytes in XCOFF32

https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.files/XCOFF.htm
https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.alangref/idalangref_und_prg_toc.htm

```
d790997dd950bb39229dc5bd3c2047ff:     file format aix5coff64-rs6000
architecture: powerpc:620, flags 0x00000177:
HAS_RELOC, EXEC_P, HAS_LINENO, HAS_SYMS, HAS_LOCALS, DYNAMIC, D_PAGED
start address 0xffffffffffffffff

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         0000c178  00000000100001f8  00000000100001f8  000001f8  2**5
                  CONTENTS, ALLOC, LOAD, RELOC, CODE
  1 .data         00002798  0000000020000370  0000000020000370  0000c370  2**4
                  CONTENTS, ALLOC, LOAD, RELOC, DATA
  2 .bss          00000048  0000000020002b08  0000000020002b08  00000000  2**3
                  ALLOC
  3 .loader       00002fad  0000000000000000  0000000000000000  0000eb08  2**3
                  CONTENTS, ALLOC, LOAD
SYMBOL TABLE:
[  0](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x00000000 pthread_self
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 10 stb 0 snstb 0
[  2](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x00000000 mmap
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 10 stb 0 snstb 0
[  4](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x00000000 munmap
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 10 stb 0 snstb 0
[  6](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x00000000 ngetpeername
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 10 stb 0 snstb 0
[  8](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x0000de00 .__strcmp64
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 7 stb 0 snstb 0
[ 10](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x0000de00 .___strcmp64
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 7 stb 0 snstb 0
[ 12](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x0000e808 .__memset64
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 7 stb 0 snstb 0
[ 14](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x0000e808 .___memset64
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 7 stb 0 snstb 0
[ 16](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x0000f400 .__memmove64
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 7 stb 0 snstb 0
[ 18](sec  0)(fl 0x00)(ty   0)(scl   2) (nx 1) 0x0000f400 .___memmove64
AUX val     0 prmhsh 0 snhsh 0 typ 0 algn 0 clss 7 stb 0 snstb 0
```

Timestamp inspection

Parsed timestamps from files (f_timdat / Offset 4 / Length 4)

```
handle.seek(4,0)
dword = handle.read(4)
timestamp = unpack("<L", dword[::-1])[0]
print strftime('\nBuild-date: %Y-%m-%d %H:%M:%S', gmtime(float(timestamp)))
```

Injection tool
d4656375180242 62c063f4a82d799a4e40ff3381014972f24ea18bc23c3b27ee → **2017-11-13 15:48:54**

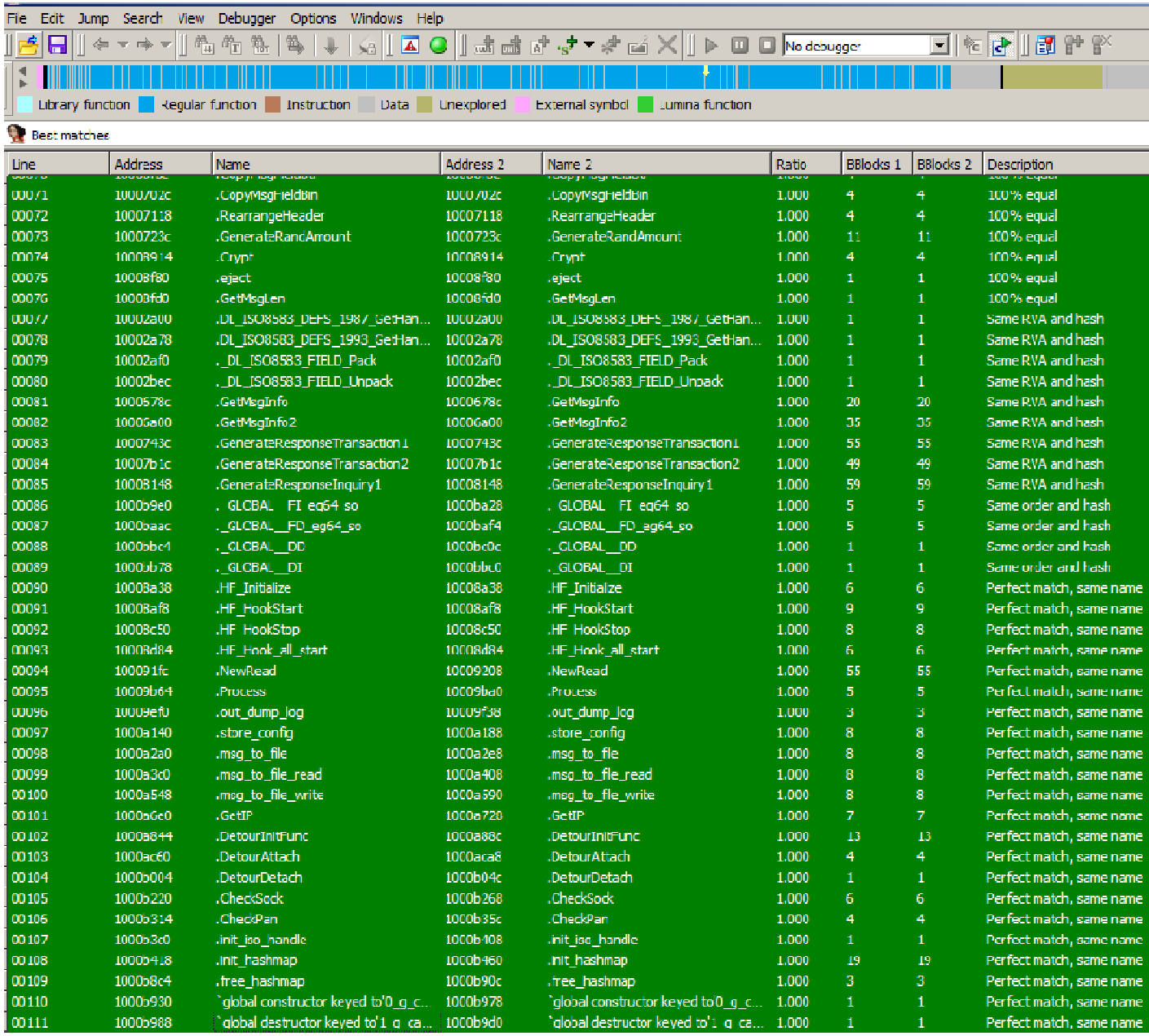
Libraries
ca9ab48d293cc84092e8db8f0ca99cb155b30c61d32a1da7cd3687de454fe86c → **2016-10-13 14:14:07**
10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba → **2017-01-13 13:34:48**
3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c → **2017-11-22 09:49:57**

First impression → Timescale between ca9ab4… and 10ac31… are 3 months.
Timestamps of d46563… and 3a5ba4 are both from November 2017
10 months newer as the other two files. Let's start bindiffing the libraries with Diaphora.

Bindiff (1)

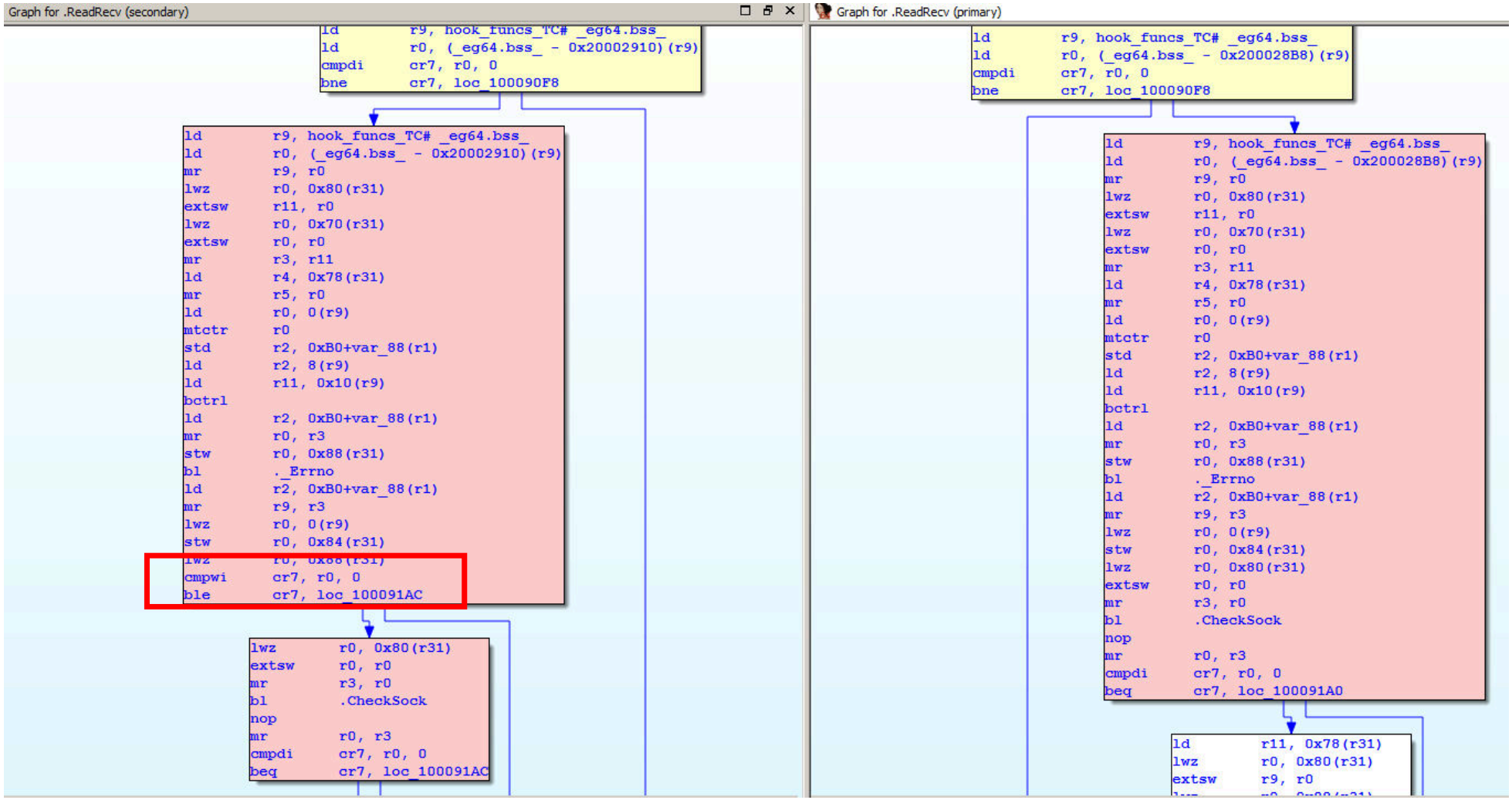
Diffing ca9ab4… against 10ac31…

Minimal differences in 3 functions
→ Partial matches vs 111 matching



Line	Address	Name	Address 2	Name 2	Ratio	BBlocks 1	BBlocks 2	Description
00071	10007 02c	.CopyMsgFieldBin	10007 02c	.CopyMsgFieldBin	1.000	4	4	100% equal
00072	10007118	.RearrangeHeader	10007118	.RearrangeHeader	1.000	4	4	100% equal
00073	1000723c	.GenerateRandAmount	1000723c	.GenerateRandAmount	1.000	11	11	100% equal
00074	10008914	.Crypt	10008914	.Crypt	1.000	4	4	100% equal
00075	10008f80	.eject	10008f80	.eject	1.000	1	1	100% equal
00076	10008fd0	.GetMsgLen	10008fd0	.GetMsgLen	1.000	1	1	100% equal
00077	10002a00	.DL_ISO8583_DEFS_1987_GetHan...	10002a00	.DL_ISO8583_DEFS_1987_GetHan...	1.000	1	1	Same RVA and hash
00078	10002a78	.DL_ISO8583_DEFS_1993_GetHan...	10002a78	.DL_ISO8583_DEFS_1993_GetHan...	1.000	1	1	Same RVA and hash
00079	10002af0	._DL_ISO8583_FIELD_Pack	10002af0	._DL_ISO8583_FIELD_Pack	1.000	1	1	Same RVA and hash
00080	10002bec	._DL_ISO8583_FIELD_Unpack	10002bec	._DL_ISO8583_FIELD_Unpack	1.000	1	1	Same RVA and hash
00081	1000678c	.GetMsgInfo	1000678c	.GetMsgInfo	1.000	20	20	Same RVA and hash
00082	10006a00	.GetMsgInfo2	10006a00	.GetMsgInfo2	1.000	35	35	Same RVA and hash
00083	1000743c	.GenerateResponseTransaction1	1000743c	.GenerateResponseTransaction1	1.000	55	55	Same RVA and hash
00084	10007b1c	.GenerateResponseTransaction2	10007b1c	.GenerateResponseTransaction2	1.000	49	49	Same RVA and hash
00085	10008148	.GenerateResponseInquiry1	10008148	.GenerateResponseInquiry1	1.000	59	59	Same RVA and hash
00086	1000b9e0	. GLOBAL FI eq64 so	1000ba28	. GLOBAL FI eq64 so	1.000	5	5	Same order and hash
00087	1000baac	._GLOBAL__FD_eq64_so	1000baf4	._GLOBAL__FD_eq64_so	1.000	5	5	Same order and hash
00088	1000bbc4	._GLOBAL__DD	1000bc0c	._GLOBAL__DD	1.000	1	1	Same order and hash
00089	1000bb78	._GLOBAL__DI	1000bbc0	._GLOBAL__DI	1.000	1	1	Same order and hash
00090	10008a38	.HF_Initialize	10008a38	.HF_Initialze	1.000	6	6	Perfect match, same name
00091	10008af8	.HF_HookStart	10008af8	.HF_HookStart	1.000	9	9	Perfect match, same name
00092	10008c50	.HF_HookStop	10008c50	.HF_HookStop	1.000	8	8	Perfect match, same name
00093	10008d84	.HF_Hook_all_start	10008d84	.HF_Hook_all_start	1.000	6	6	Perfect match, same name
00094	100091fc	.NewRead	10009308	.NewRead	1.000	55	55	Perfect match, same name
00095	10009b64	.Process	10009ba0	.Process	1.000	5	5	Perfect match, same name
00096	10009ef0	.out_dump_log	10009f38	.out_dump_log	1.000	3	3	Perfect match, same name
00097	1000a140	.store_config	1000a188	.store_config	1.000	8	8	Perfect match, same name
00098	1000a2a0	.msg_to_file	1000a2e8	.msg_to_file	1.000	8	8	Perfect match, same name
00099	1000a3c0	.msg_to_file_read	1000a408	.msg_to_file_read	1.000	8	8	Perfect match, same name
00100	1000a548	.msg_to_file_write	1000a390	.msg_to_file_write	1.000	8	8	Perfect match, same name
00101	1000a6c0	.GetIP	1000a720	.GetIP	1.000	7	7	Perfect match, same name
00102	1000a844	.DetourInitFunc	1000a88c	.DetourInitFunc	1.000	13	13	Perfect match, same name
00103	1000ac60	.DetourAttach	1000aca8	.DetourAttach	1.000	4	4	Perfect match, same name
00104	1000b004	.DetourDetach	1000b04c	.DetourDetach	1.000	1	1	Perfect match, same name
00105	1000b220	.CheckSock	1000b268	.CheckSock	1.000	6	6	Perfect match, same name
00106	1000b314	.CheckPan	1000b35c	.CheckPan	1.000	4	4	Perfect match, same name
00107	1000b3c0	.init_iso_handle	1000b408	.init_iso_handle	1.000	1	1	Perfect match, same name
00108	1000b418	.init_hashmap	1000b460	.init_hashmap	1.000	19	19	Perfect match, same name
00109	1000b8c4	.free_hashmap	1000b90c	.free_hashmap	1.000	3	3	Perfect match, same name
00110	1000b930	`global constructor keyed to'0_g_c...	1000b978	`global constructor keyed to'0_g_c...	1.000	1	1	Perfect match, same name
00111	1000b988	`global destructor keyed to'1_g_ca...	1000b9d0	`global destructor keyed to'1_g_ca...	1.000	1	1	Perfect match, same name

Bindiff (1) → Example compare in function ReadRecv()



Bindiff (2)

Diffing ca9ab4... against 3a5ba4...

56 Best Matches, 36 Partial Matches, 13 unmatched primary, 14 unmatched secondary
Seems to be same origin, but newer library 3a5ba4... looks heavily reengineered.

ca9ab4...

Unmatched in primary

Line	Address	Name
00000	100066b4	.SkipMsgHeader
00001	1000678c	.GetMsgInfo
00002	10006a00	.GetMsgInfo2
00003	10006f5c	.CopyMsgFieldStr
00004	1000702c	.CopyMsgFieldBin
00005	10007118	.RearrangeHeader
00006	1000743c	.GenerateResponseTransaction1
00007	10007b1c	.GenerateResponseTransaction2
00008	10008148	.GenerateResponseInquiry1
00009	10009208	.NewRead
00010	10009de0	.NewWrite
00011	1000a408	.msg_to_file_read
00012	1000a590	.msg_to_file_write
00013	1000b268	.CheckSock

3a5ba4...

Unmatched in secondary

Line	Address	Name
00000	10000220	.__init_aix_libgcc_cxa_atexit
00001	10000250	`global destructor keyed to'65535_0___dso_handle
00002	10000254	.strtold
00003	10000ab0	.NewRecv
00004	100011e8	.NewSend
00005	1000187c	.msg_to_file_recv
00006	10001aa4	.msg_to_file_send
00007	100027f4	.BlacklistCheck
00008	10003af8	.DL_ASCHEX_TO_UINT32
00009	100043a4	.DL_UINT32_TO_ASCHEX
00010	10009bb0	.DL_MAKE_MSGINFO
00011	1000a020	.DL_MAKE_RESPONSE_MSG
00012	1000a75c	.DL_MAKERESPOND
00013	1000a8b4	.DL_GET_PAN_STR
00014	1000ab64	.GetMsgFields

AIX PowerPC Assembly 101 (Register types)

Before diving deeper into the dissection of the XCOFF binaries and for better understanding let me give you a very basic introduction to AIX PowerPC assembly.

- PPC is a RISC architecture
- 32 (0-31) General Purpose Registers (R0-R31)
- 32-Floating Pointer Registers
- Special Purpose Registers
  - PC/IAR → holds address to be executed next
  - LR → holds address of procedure for branching
  - CR → 8 (0-7) conditional registers, divided into 4 bit fields, holding result of compare instruction
  - CTR → count register
  - XER → exception register (overflows, carry conditions)
  - FPSCR → floating pointer status and control register

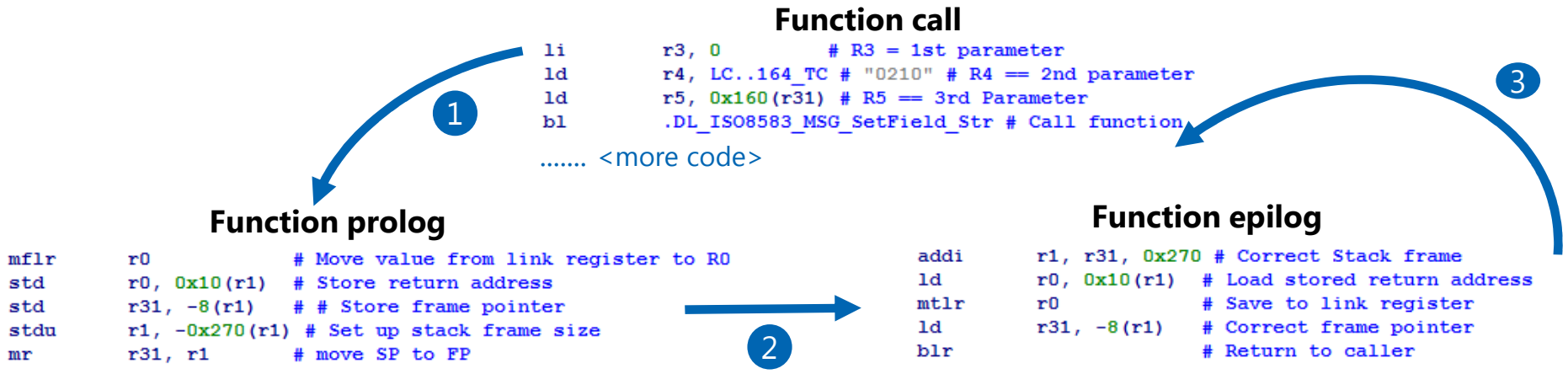
AIX PowerPC Assembly 101 (Instructions)

- Fixed length 32-Bit instructions
  - Opcode: 6 bits
  - Source register: 5 bits
  - Destination register: 5 bits
  - Immediate value: 16 bits

- Instructions Set consists of:
  - Integer instructions (ADDI, LI, SUBF, MULLH, DIV, CMPI, ORI …)
  - Floating point instructions (FMR, LFS, STFS…)
  - Load and store instructions (LWZ, STW, MFLR, MTLR …)
  - Branch and flow control instructions (BEQ, BNE, BGT, BL, BLR …)
  - Various instructions (CRXOR, CLRLWI …)

AIX PowerPC Assembly 101 (Most important for current needs)

- Function parameters are passed through registers
  - R3-R10 → Parameters 1-8
- When function completes return code value is stored in R3
- Branch/Call to a function via BL instruction
- Return from a function via BLR instruction
- R1 holds the stack pointer
- R31 holds the frame pointer
- R0 → mostly used as temporary scratch register



Hidden Cobra's Fast Cash injection tool

Hash → d465637518024262c063f4a82d799a4e40ff3381014972f24ea18bc23c3b27ee

- The technique used by the attackers to inject code into a running application is quite rare, because of its complexity.
- By accessing directly the PROCFS via /proc/<PID>/mem they get access to the state and memory of each process, to read/write arbitrary data to it.
- To access the PROCFS standard system calls are used like open(), close(), pread(), write().
- Directory **ctl** (control) is a write only directory, which holds the process control file and can be used to send control messages, e.g. to start process, stop on specific events etc.
- Directory **as** (address space) gives access to the process address space mappings.
- Directory **status** holds general status and state information about a process.

```
addi      r9, r2, (g_pid - 0x20000930)
lwz       r9, (g_pid - 0x20000630)(r9)
extsw     r9, r9
addi      r10, r31, 0x70 # 'p'
mr        r3, r10
ld        r4, LC..41_TC # "/proc/%d/ctl"
mr        r5, r9
bl        .sprintf
ld        r2, 0x390+var_368(r1)
addi      r9, r2, (g_pid - 0x20000930)
lwz       r9, (g_pid - 0x20000630)(r9)
extsw     r9, r9
addi      r10, r31, 0x170
mr        r3, r10
ld        r4, LC..43_TC # "/proc/%d/status"
mr        r5, r9
bl        .sprintf
ld        r2, 0x390+var_368(r1)
addi      r9, r2, (g_pid - 0x20000930)
lwz       r9, (g_pid - 0x20000630)(r9)
extsw     r9, r9
addi      r10, r31, 0x270
mr        r3, r10
ld        r4, LC..45_TC # "/proc/%d/as"
mr        r5, r9
bl        .sprintf
ld        r2, 0x390+var_368(r1)
addi      r9, r31, 0x70 # 'p'
mr        r3, r9
lis       r4, 0x400
ori       r4, r4, 1 # 0x4000001
bl        .open
```

Hidden Cobra's Fast Cash injection tool

Hash → d465637518024262c063f4a82d799a4e40ff3381014972f24ea18bc23c3b27ee

```
bash-4.3# ./injection
Usage: injection pid dll_path mode [handle func toc]
       mode = 0 => Injection
       mode = 1 => Ejection
```

- Process Injection step by step:
  - Attach to process and inject shared object
  - Suspend process
  - Store important process data for later ejection
  - Modify instruction address register (IAR) to point to the new code
  - Use previous IAR as return address
  - Resume process to start shared object
    See → http://uw714doc.sco.com/en/man/html.4/proc.4.html (Section → Control Messages for better understanding messages like PCSET, PCSTOP, PCTRACE, PCSENTRY, PCRUN etc.)
  - Example code to illustrate usage → https://github.com/openzfs/openzfs/blob/master/usr/src/cmd/sgs/librtld_db/demo/common/main.c

```
ld      r3, LC..126_TC # "[main] Inject Start"
bl      .out_log
bl      .proc_attach
li      r3, 0
bl      .proc_continue
li      r3, 0
li      r4, 0
bl      .proc_wait
ld      r3, LC..128_TC # "[main] SAVE REGISTRY"
bl      .out_log
addi    r9, r31, 0x98
mr      r3, r9
bl      .proc_getregs
addi    r9, r31, 0x98
mr      r3, r9
bl      .out_regs
addi    r8, r31, 0x218
addi    r10, r31, 0x98
li      r9, 0x180
mr      r3, r8
mr      r4, r10
mr      r5, r9
bl      .memmove
nop
ld      r9, 0x218(r31)
addi    r9, r9, -0x10
mr      r3, r9
li      r4, 0x4000
```

Setting code hooks within the target payment switch application

Hash → 3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c

- After load of shared object file:
  - Constructor _GLOBAL__I... starts the HF_Hook_all_start() function
- Then several initialize functions are being called, e.g.:
  - Store config file /tmp/.ICE-unix/config_%d
  - Open /tmp/.ICE-unix/applist.dat with an encrypted list of attacker controlled Primary Accounts Numbers (PAN) → used later in CheckPAN() function
  - Init ISO8583 definitions handler (1987) (International standard for financial transaction card originated interchange messaging)

```
.HF_Hook_all_start:                     # CODE XREF: `global constr

.set sender_sp, -0x80
.set var_8, -8
.set sender_lr,  0x10

                mflr    r0
                std     r0, sender_lr(r1)
                std     r31, var_8(r1)
                stdu    r1, sender_sp(r1)
                mr      r31, r1
                ld      r3, LC..11_TC # "Load"
                bl      .out_dump_log
                nop
                addi    r9, r2, (off_20001D28+0x68 - 0x20002678)
                ld      r10, (off_20001D28+0x68 - 0x20001D90)(r9)
                ld      r9, eject.P8_TC # 0x20001D98
                ld      r9, (off_20001D28+0x70 - 0x20001D98)(r9)
                ld      r3, LC..15_TC # "func=%llX, toc=%llX"
                mr      r4, r10
                mr      r5, r9
                bl      .out_dump_log
                nop
                addi    r3, r2, (off_20001D28+0x68 - 0x20002678)
                li      r4, 0x10
                bl      .store_config
                nop
                bl      .init_hashmap
                nop
                mr      r9, r3
                mr      r10, r9
                li      r9, -1
                cmpw    cr7, r10, r9
                bne     cr7, loc_10000764
                li      r9, 0
                b       loc_100007B8
# ---------------------------------------------------------------------------

loc_10000764:                           # CODE XREF: .HF_Hook_all_s
                ld      r3, LC..17_TC # "init_hashmap succ"
                bl      .out_dump_log
                nop
                bl      .init_iso_handle
```

Setting code hooks within the target payment switch application

Hash → 3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c

- HF_Initialize() function then triggers:
  - DetourInitFunc() to detour recv() and send() from libc.a(shr_64.o) to NewRecv() and NewSend() to control all incoming/outgoing data within the payment switch application.
    Note → Samples ca9ab4… and 10ac31… hook functions read() and write() with NewRead() and NewWrite()
  - Original function pointers get replaced by new ones in the ToC
- HF_HookStart() function calls → DetourAttach()
- Unhooking is done in Destructor _GLOBAL__D…
  - Call to HF_Hook_all_stop() function → calling HF_HookStop() → DetourDetach() to remove hooks

```
bl        .HF_Initialize
nop
mr        r9, r3
cmpdi     cr7, r9, 0
bne       cr7, loc_100007AC
ld        r3, LC..21_TC # "HF_Initialize failed"
bl        .out_dump_log
nop
li        r9, 0
b         loc_100007B8
-----------------------------------------------

                        # CODE XREF: .HF_Hook_al
bl        .HF_HookStart


.HF_Hook_all_stop:                      # CODE XRE

.set sender_sp, -0x80
.set var_8, -8
.set sender_lr,  0x10

                mflr      r0
                std       r0, sender_lr(r1)
                std       r31, var_8(r1)
                stdu      r1, sender_sp(r1)
                mr        r31, r1
                ld        r3, LC..23_TC # "Unload"
                bl        .out_dump_log
                nop
                bl        .HF HookStop
```

Comparing encryption keys in applist.dat crypt() function

Libraries with high similarity use same crypto key:
Hash → 3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c
Key → zRuaDglxjec^tDttSlsklqc^mSlsklqc^mNgq`lyznqr[q^123

Newer library uses a different one:
Hashes → ca9ab48d293cc84092e8db8f0ca99cb155b30c61d32a1da7cd3687de454fe86c
10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba
Key → dkfjy)1*290(yY89!(p#y!@pURH1;2EH89Pu*KF(p3RHH89

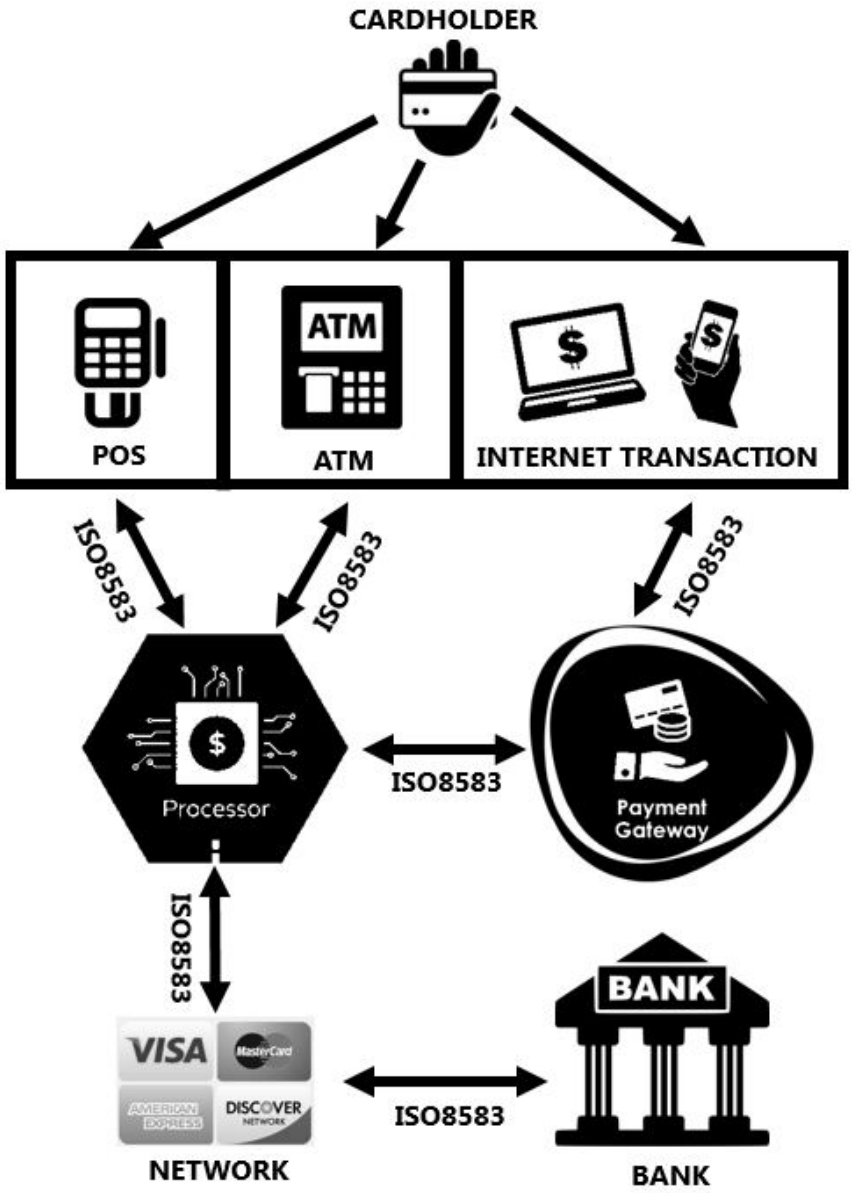
Another indicator samples are from at least two different operations

```
.Crypt:                                 # CODE XREF: .Process+298↑p
                                        # .init_hashmap+1AC↑p
                                        # DATA XREF: ...

.set sender_sp, -0x60
.set var_8, -8

                std     r31, var_8(r1)
                stdu    r1, sender_sp(r1)
                mr      r31, r1
                std     r3, 0x90(r31)
                mr      r9, r4
                stw     r9, 0x98(r31)
                li      r9, 0
                stw     r9, 0x30(r31)
                li      r9, 0
                std     r9, 0x38(r31)
                li      r9, 0
                std     r9, 0x40(r31)
                li      r9, 0
                stw     r9, 0x48(r31)
                ld      r9, LC..190_TC # "zRuaDglxjec^tDttSlsklqc
                ld      r8, (aZruadglxjecTdt - 0x1000C130)(r9) #
                ld      r10, (aZruadglxjecTdt+8 - 0x1000C130)(r9)
                lwz     r9, (aZruadglxjecTdt+0x10 - 0x1000C130)(r
                std     r8, 0x38(r31)
                std     r10, 0x40(r31)
                stw     r9, 0x48(r31)
                li      r9, 0
                stw     r9, 0x30(r31)
                b       loc_1000AE90
```

ISO8583 standard basics

- ISO8583 is an international standard for systems exchanging electronic transactions initiated by cardholders using credit/debitcards.
- Each time customers use cards at POS terminals or ATMs for payment, ISO8583 is used at a specific point in the communication chain for such transactions.
- Especially payment card companies like Mastercard, VISA networks and other institutions base their authorization communications on ISO8583.
- Although ISO8583 defines many standard data elements, remaining the same in all networks, a few additional fields can be used for custom usage to pass network specific details.
- An ISO 8583 message consists of three parts:
  - Message type identifier (MTI)
  - Bitmaps (one or more) indicating present data elements
  - Data elements → fields containing the actual information of a message

https://www.nibss-plc.com.ng/images/api/POS_Interface_Specification_ver_1.11.pdf
https://www.codeproject.com/Articles/100084/Introduction-to-ISO



ISO8583 library used by actors

Oscar Sanderson's ISO8583 C-Library

http://www.oscarsanderson.com/iso-8583/

https://github.com/sabit/Oscar-ISO8583/

Functions:

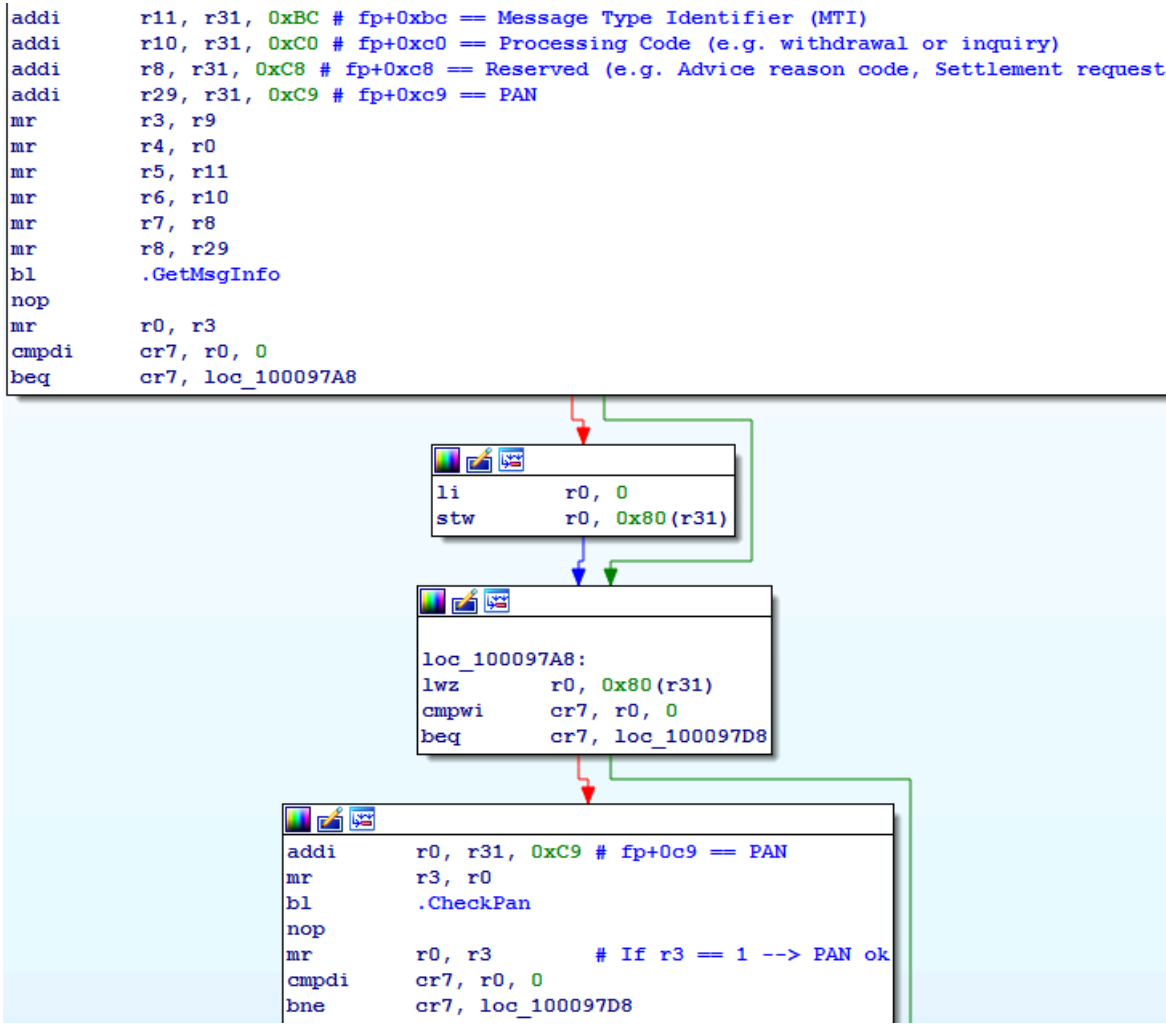
- DL_ISO8583_DEFS_1993_GetHandler
- DL_ISO8583_MSG_Init
- DL_ISO8583_MSG_Unpack
- DL_ISO8583_MSG_SetField_Str
- DL_ISO8583_MSG_SetField_Bin
- DL_ISO8583_MSG_GetField_Str
- DL_ISO8583_MSG_GetField_Bin
- DL_ISO8583_MSG_Pack
- DL_ISO8583_MSG_Free
- DL_ISO8583_MSG_RemoveField
- ……

```
li        r3, 0
li        r4, 0
mr        r5, r0
bl        .DL_ISO8583_MSG_Init
nop
addi      r0, r31, 0x8B8
li        r3, 0
li        r4, 0
mr        r5, r0
bl        .DL_ISO8583_MSG_Init
nop
lwz       r0, 0x10D8(r31)
clrldi    r0, r0, 32
addi      r9, r31, 0x98
ld        r3, isoHandler_TC # isoHandler
ld        r4, 0x88(r31)
mr        r5, r0
mr        r6, r9
bl        .DL_ISO8583_MSG_Unpack
nop
mr        r0, r3
std       r0, 0x90(r31)
ld        r0, 0x90(r31)
cmpdi     cr7, r0, 0
bne       cr7, loc_100075A0
ld        r0, Authorization_Response # "0110"
addi      r9, r31, 0x8B8
li        r3, 0
mr        r4, r0
mr        r5, r9
bl        .DL_ISO8583_MSG_SetField_Str
```

ISO8583 incoming request message (MTI 100) parsing

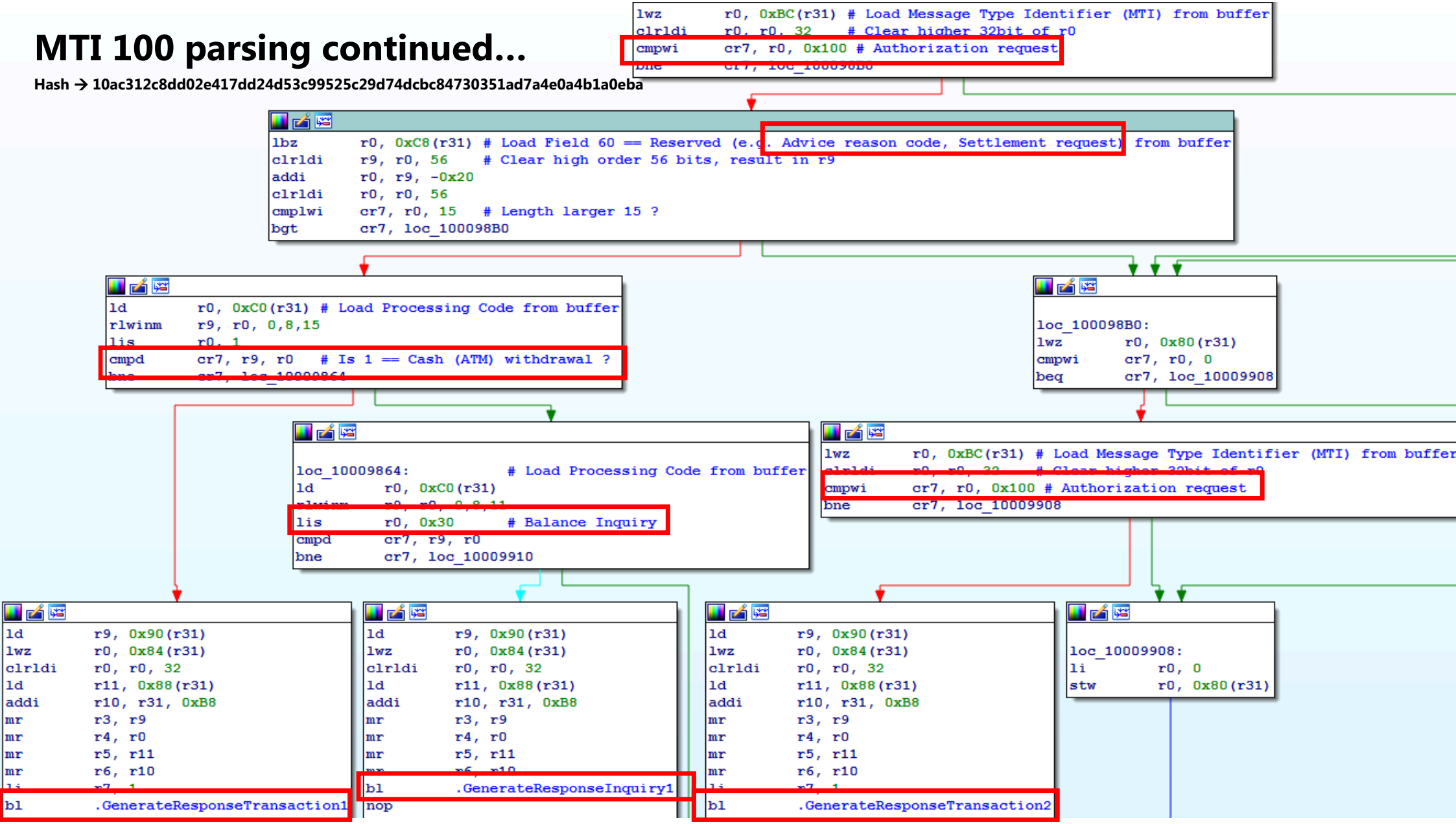
Hash → 10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba

- Read 4 ISO8583 data fields from incoming request message
  - Message Type Identifier (MTI)
  - Processing Code
  - Reserved (e.g. Advice reason code)
  - Primary Account Number (PAN)
- CheckPan() → compare PAN from message against attacker controlled PAN hash list



MTI 100 parsing continued…

Hash → 10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba



ISO8583 (MTI 110) response messages (Data field/Type)

Hash → 10ac312c8dd02e417dd24d53c99525c29d74dcbc84730351ad7a4e0a4b1a0eba

GenerateResponseTransaction1

0	Message Type Indicator (MTI)
2	Primary account number (PAN)
3	Processing Code
4	Amount, transaction
7	Transmission date/time
11	System trace audit number (e.g. transaction no.)
14	Expiration date
19	Acquiring institution (country code)
22	Point of service entry mode
25	Point of service condition code
32	Acquiring institution identification code
35	Track 2 data
37	Retrieval reference number
38	Authorization identification response
39	Response code (e.g. 00=Approved, 51=Non-sufficient funds, 55=invalid PIN)
41	Card acceptor terminal identification
42	Card acceptor identification code
44	Additional response data (e.g. phone no.)
49	Currency code, transaction
62	Reserved (private) (e.g. invoice number, INF Data etc.
63	Network data (Financial Network code, Banknet Ref. No., Banknet No. etc.)

GenerateResponseTransaction2

0	Message Type Indicator (MTI)
2	Primary account number (PAN)
3	Processing Code
4	Amount, transaction
7	Transmission date/time
11	System trace audit number (e.g. transaction nr.)
19	Acquiring institution (country code)
25	Point of service condition code
32	Acquiring institution identification code
37	Retrieval reference number
38	Authorization identification response
39	Response code (e.g. 00=Approved, 51=Non-sufficient funds, 55=invalid PIN)
41	Card acceptor terminal identification
42	Card acceptor identification code
44	Additional response data (e.g. phone no.)
49	Currency code, transaction
62	Reserved (private) (e.g. invoice number, INF Data etc.
63	Network data (Financial Network code, Banknet Ref. No., Banknet No. etc.)

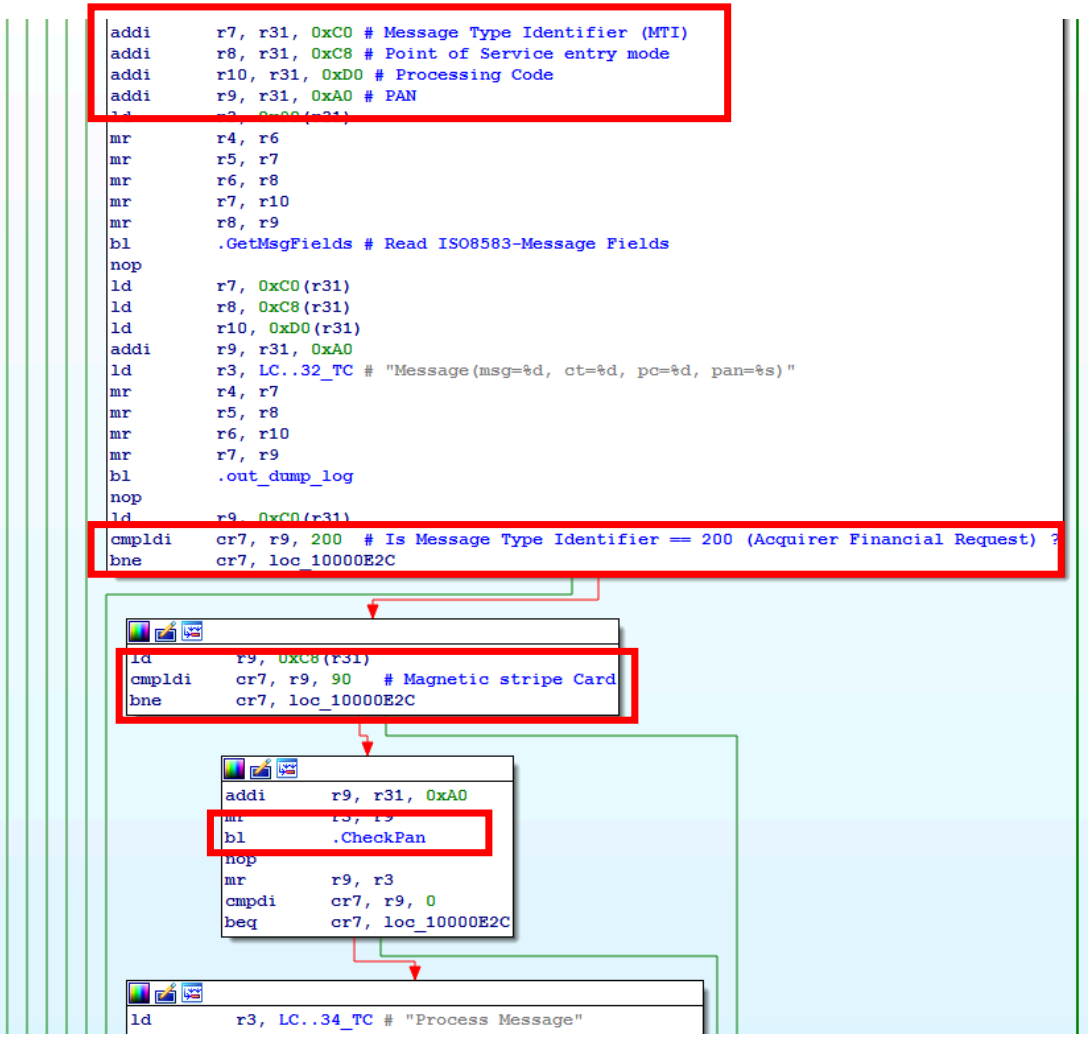
GenerateResponseInquiry1

0	Message Type Indicator (MTI)
2	Primary account number (PAN)
3	Processing Code
7	Transmission date/time
11	System trace audit number (e.g. transaction no.)
14	Expiration date
18	Merchant type, or merchant category code
19	Acquiring institution (country code)
22	Point of service entry mode
25	Point of service condition code
32	Acquiring institution identification code
35	Track 2 data
37	Retrieval reference number
38	Authorization identification response
39	Response code (e.g. 00=Approved, 51=Non-sufficient funds, 55=invalid PIN)
41	Card acceptor terminal identification
42	Card acceptor identification code
44	Additional response data (e.g. phone no.)
49	Currency code, transaction
54	Additional amounts (Account balance)
62	Reserved (private) (e.g. invoice number, INF Data etc.)
63	Network data (Financial Network code, Banknet Ref. No., Banknet No. etc.)

ISO8583 incoming request message (MTI 200) parsing

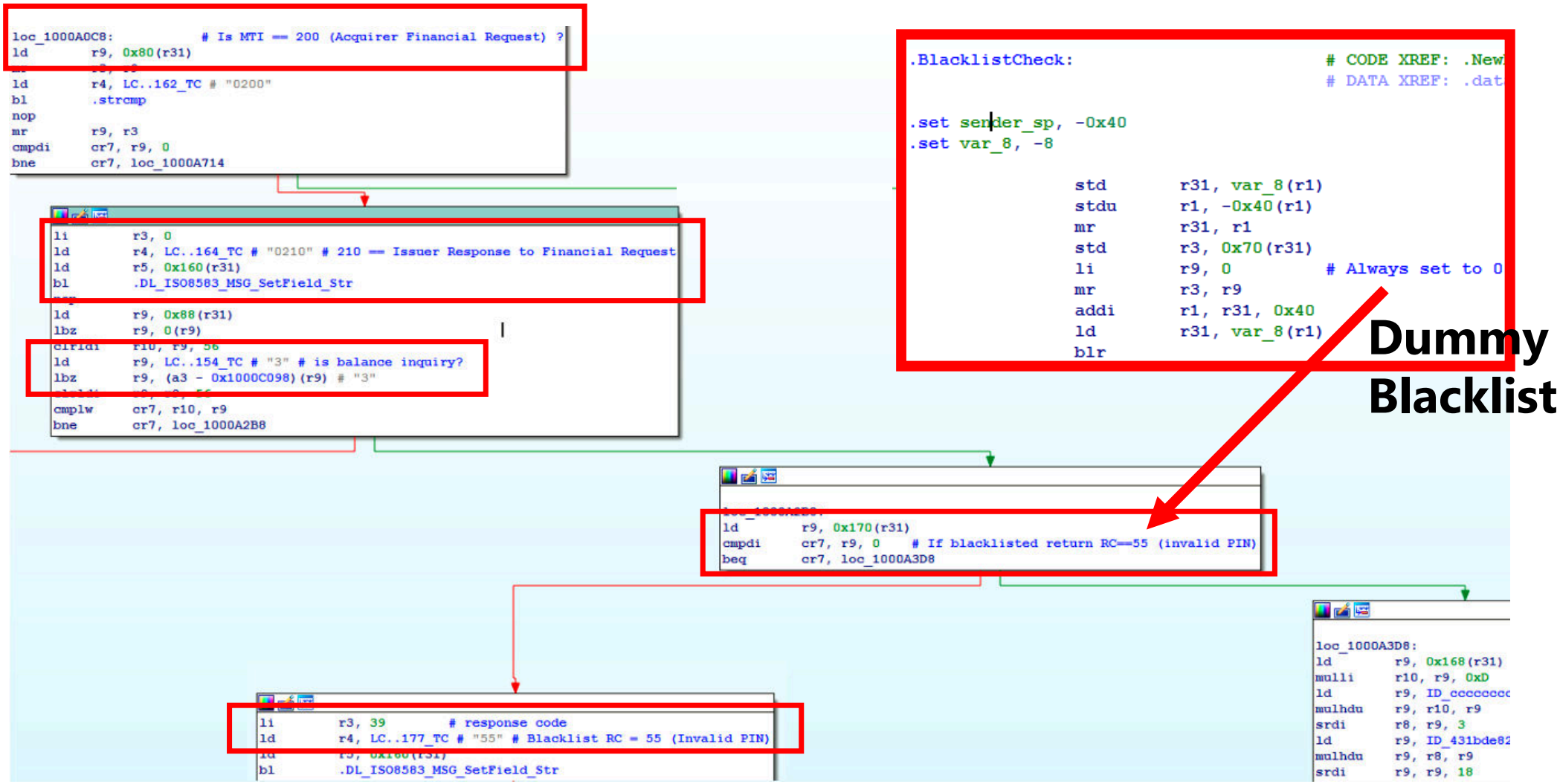
Hash → 3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c

- Read 4 ISO8583 data fields from incoming request message
  - Message Type Identifier (MTI)
  - Point of Service entry mode
  - Processing Code
  - Primary Account Number (PAN)
- Check if Point of Service entry mode is 90 (Magstripe)
- CheckPan() → compare PAN from message against attacker controlled PAN hash list



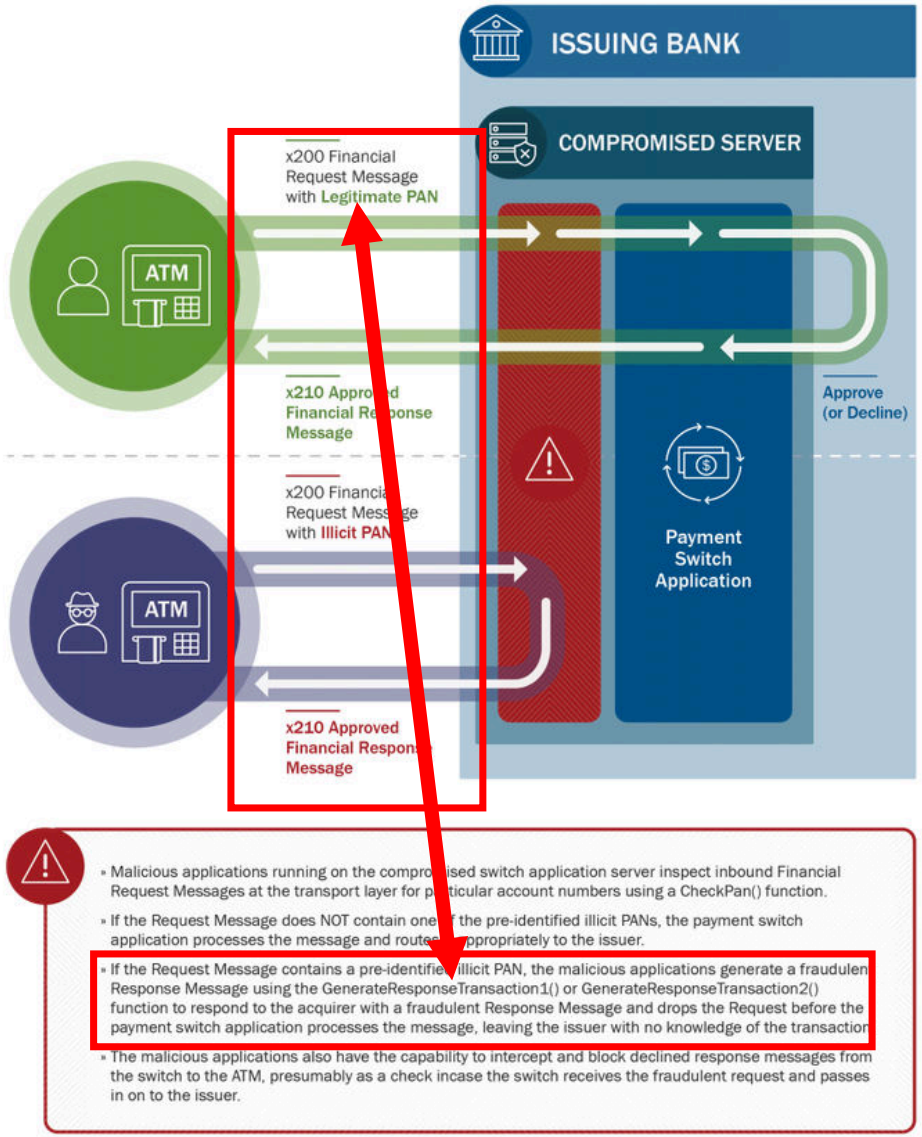
ISO8583 MTI 200 requests parsing continued and 210 responses

Hash → 3a5ba44f140821849de2d82d5a137c3bb5a736130dddb86b296d94e6b421594c



Conclusions so far…

- Library hashes ca9ab4… + 10ac31… parse MTI 100 messages, while 3a5ba4… parses MTI 200 messages. Moreover different reserved ISO8583 fields are used, suggesting network specific use. Two more indicators samples are from at least two different operations.
- Taking these observations into account, the US-CERT report is a little bit misleading here as functions GenerateResponse* belong to samples parsing MTI 100 messages and not MTI 200.



Source → https://www.us-cert.gov/ncas/alerts/TA18-275A

So what payment switch application was attacked?

- Wondering what kind of payment switch software was attacked at the Cosmos Bank in India, i did some OSINT…
- According to news at Indian Cooperative from October 13th 2015 the Cosmos Bank purchased a new ATM switch called SmartVista from BPC Banking Technologies which has its headquarters in Switzerland.
  http://www.indiancooperative.com/co-op-news-snippets/cosmos-bank-yet-again-goes-high-tech/
- In another news at ATMMarketPlace BPC said in a press Cosmos will roll out secure e-commerce in 2015 followed by a full launch of the platform in the first quarter of 2016.
  https://www.atmmarketplace.com/news/cosmos-expands-atm-payments-capabilities-with-smartvista/

Gaining more insights about SmartVista

To get more certainty let's check if SmartVista supports AIX and ISO8583.
An IBM Redbook from 2008 helps us here →
http://www.redbooks.ibm.com/abstracts/redp4373.html?Open

4.1.1 SmartVista technical and physical architecture on System i

IBM System i servers have proved to be reliable, high-performance, cost-effective systems that are safe from hackers and viruses. The System i hardware platform ensures performance and reliability, creating an environment for the deployment of various business applications from basic financial applications and payment systems to internet banking. In its turn, the IBM i5/OS® operating system provides the environment for integrating the information resources of banks and non-financial organizations, which helps reduce the total cost of ownership (TCO) of IT resources and helps increase their efficiency and fault-tolerance.

The System i software/hardware platform is based on open standards so that banks and financial institutions can use their existing system resources instead of buying a new server for each application. The platform concurrently supports applications running on i5/OS, Microsoft Windows, Linux, and IBM AIX 5L environments. System i integrates the DB2® database management system, security and performance features, LAN and Internet connectivity, backup and restore functionality, report generation, and file and print management functions. DBMS integration in the operating system results in high performance, reliability, and management simplicity.

SmartVista Front-End

SmartVista Front-End is a lean and mean real-time transaction engine that can be scaled up or out as needed through unlimited process parallelization capabilities and efficient communications infrastructure.

SmartVista Front-End handles all online real-time interactions and activities that are required to provide fast and error-free authorization processing at both the issuer and acquirer sides. Typically, it processes all transaction origination devices, online payment scheme interfaces, online host interfaces, Internet and wireless interfaces, and any other online interfaces.

The key functional areas that are supported by SmartVista Front-End include:

- Authorization requests capture, processing and routing
- Stand-in, host, and serial multi-host authorization
- ATM and POS network management and monitoring, including support for multifunctional self-service kiosks
- Online transaction monitoring using rule-based or statistics models to detect fraudulent transactions

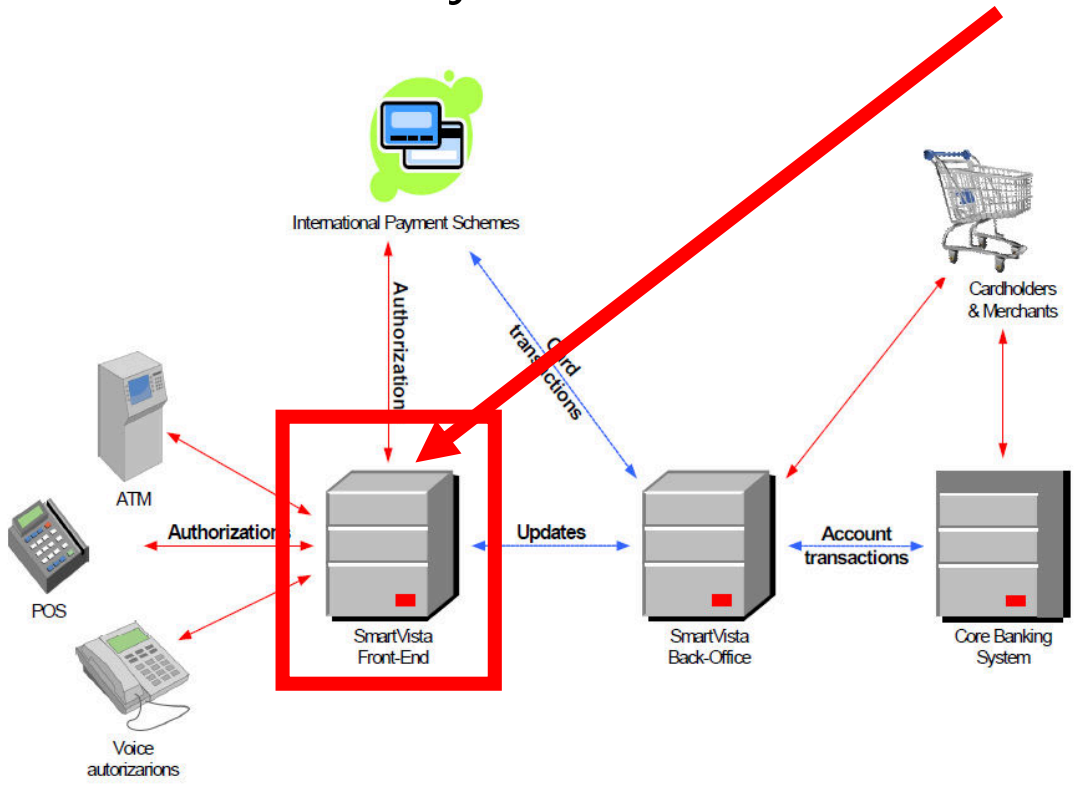
SmartVista i is a result of extensive collaboration between the BPC research and development team and IBM experts from Russia, the United Kingdom, and the U.S. Built around the growing demands of banks and interbank processors in both mature and emerging markets, the SmartVista i solution represents a reliable and high-performance

ATM transaction acquiring

SmartVista i supports most ATM application protocols. These include NDC+, Diebold 912/DDC, and generic ISO 8583.

BPC SmartVista architecture (according to IBM Redbook)

So if we assume a SmartVista payment switch was attacked it seems likely the malicious code was injected at the SmartVista Front-End server.



Source → http://www.redbooks.ibm.com/abstracts/redp4373.html?Open

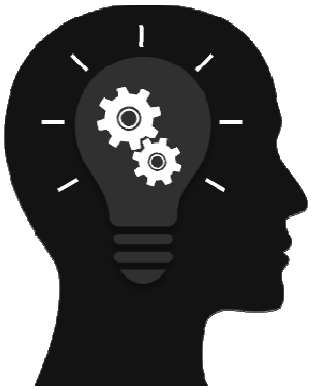
Missing response message integrity (MACing)

According to the US-CERT report actors enabled cash to be simultaneously withdrawn from ATMs in 23 different countries.

- Taking this information into account, a lot of people asked how attackers bypassed the Message Authentication Code (MAC) validation.
  - For better understanding → It's best practice that most ATMs and PoS devices check the integrity of a message between the sender and the receiver, by generating a MAC (ISO8583 data element 64). Most ATMs and PoS devices reject a transaction if the MAC validation fails.
  - None of the malicious libraries sets field 64.
- Without having ISO8583 trace messages from the heist at the Cosmos Bank the mystery can't be answered with certainty.
  - Assumption → Another part/or function of the payment switch has taken over the MACing, before the ISO8583 response messages left the bank's network.

Final thoughts

- Rumors have it that the attackers had been in the bank's network, staying under the radar for many weeks before the final heist occurred between 11th and 13th August 2018.
  - This is not surprising, as even experienced actors need time to reach and manipulate their final target, especially if the architecture and protocols of the target are proprietary and/or network specific like in this case.
- This heist shows once more → Sophisticated attackers reach their goal, regardless of operating system or application platform.
- Attacks of this magnitude can be significantly mitigated or prevented if suitable security measures are implemented.
  - US-CERT lists a whole bunch of recommendations to mitigate and detect similar attacks → **https://www.us-cert.gov/ncas/alerts/TA18-275A**



Thanks!

@r3c0nst