An Open-source Cryptographic Coprocessor

Peter Gutmann

University of Auckland, New Zealand

Problems with Crypto on End-user Systems

Passive attack

- `ReadProcessMemory`
- Subclass Windows shell, hook apps on startup
- Patch systemwide user-to-kernel mode jump table
- AppInitDLLs registry key causes DLL to be loaded and called on app startup
- Unix: `ptrace` with PTRACE_ATTACH

Active attack

- `SuspendThread`/`VirtualProtectEx`/`WriteProcessMemory`/ `ResumeThread`

Assisted attack

- Notification Packages registry key hands over all passwords
- ExpoOffload registry key hands over all private keys

Avoiding the Problem

Unix: Run as a dæmon

Windows NT: Run as a service

Windows 95: Run away

But

- All NT services run under the shared system account
- Load a new service dynamically and use `ReadProcessMemory` on other services
- Overwrite parent process handle with that of system account

Why End-user OS's will Never be Secure

Consumers don't (really) care about security

- 92% of Fortune 1000 managers were worried about ActiveX, Java, etc etc
- About three quarters allowed them into their internal networks anyway
- About half didn't even scan for them

Comments from security experts

- Which sells more products, really secure software or really easy-to-use software?
- Corporate cultures are focused on money, not product
- The way to win is to design software that is as insecure as you can possibly get away with […] Users prefer cool features to security

Why End-user OS's will Never be Secure (ctd)

Most bugs will never be fixed

- 1/3 of faults have MTTF of ~5000 years
- 1/3 of faults have MTTF of ~1500 years
- 2% have MTTF of 5 years → For marketing purposes, remove only this 2%

Apps are used in stereotyped ways which exercise only a tiny portion of their code

- Removing visible defects will keep most users happy
- "It crashes when you do *X*? Don't do that then"

Users are forced to use insecure software

- Businesses need to handle Word and Excel documents, web pages loaded with ActiveX and JavaScript in order to operate

Solving the Problem

Standard approach

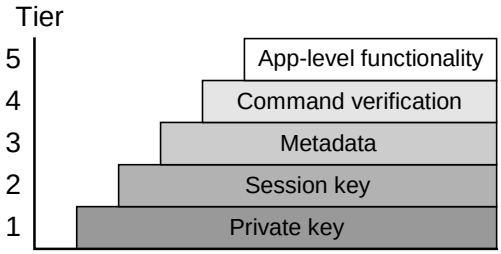
- Move the insecurity away from the crypto
- Requires a secure OS (Orange Book B2 minimum)

Mohammed and the mountain approach

- Move the crypto away from the insecurity

Coprocessor Design Issues

How much functionality should we move across?



Tier 1

- Private key protection only (smart card)
- All operations are controlled by untrusted host
  - Can request decryption or signing of anything
- Barely better than no protection at all

Coprocessor Design Issues (ctd)

Tier 2

- Session key + key wrap operations (Fortezza)
- No cryptovariables are present on untrusted host
- Device is still controlled by untrusted host
  - Fortezza protocols like CSP/MSP include complex security mechanisms, but enforcement is left to the host (!!)

Tier 3

- All data processing + metadata control
- Host can request encryption or signing of entire message
- Coprocessor performs message formatting, adds timestamp and signer identity, etc

Coprocessor Design Issues (ctd)

Tier 4

- Command verification
- Trusted I/O channel to allow user to confirm commands from host
  - "Do you really want to sign this?"

Tier 5

- Application-level functionality
- Needs to have message viewer, editor, MUA, …
  … MIME attachments, HTML, JavaScript, ActiveX, …
- Coprocessor now needs its own coprocessor for security

Best tradeoff is tier 3 or tier 4 coprocessor

Coprocessor Hardware

Standard approach

- ASICs, microcontrollers, custom hardware

COTS approach

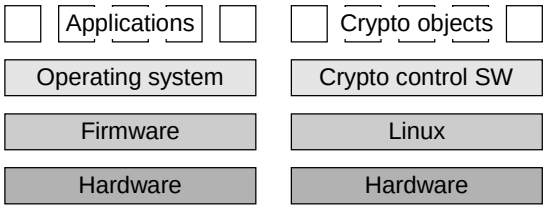
- PC/104 embedded PC
- Biscuit PC
- SIMM PC

Tier 1-3 crypto processor hardware

- Smart card: 5MHz 8 bit CPU, 256 bytes RAM, 4K EEPROM
- Fortezza card: 10/20MHz ARM CPU, 64kB RAM, 128kB EEPROM
- Open-source copro: 133MHz Pentium CPU, 16MB RAM, 8MB flash

Coprocessor Firmware

Redefine role of various system layers for crypto-specific functionality

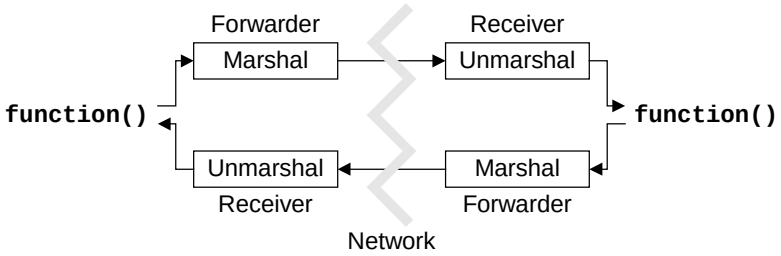


Best choice is embedded Linux

- Drivers for every imaginable piece of hardware
- Acts as bootstrap loader and resource manager for crypto control software
- Free/open source

Crypto Functionality Implementation

Data is moved to/from coprocessor using forwarder/receiver mechanism

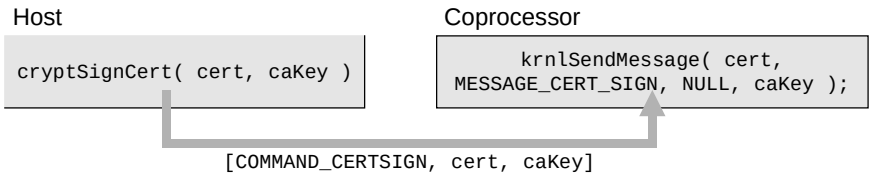


Communications options

- Ethernet
- USB
- Parallel port (EPP/ECP)
- Carpet static

Crypto Functionality Implementation (ctd)

Crypto-related function calls on local system are forwarded to coprocessor for processing

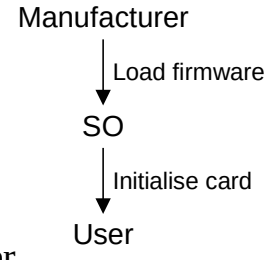


Host sees only standard software crypto interface

Coprocessor Session Management

Tier 2 processors have relatively sophisticated session control

- Manufacturer initialises device
- Security officer (SO) loads security parameters
- User uses device



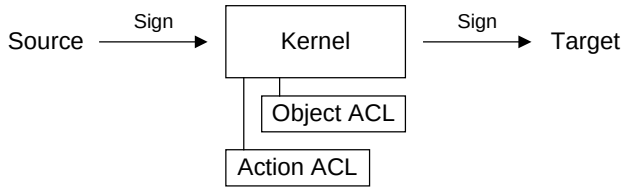
SO functions can't be performed by user
User functions can't be performed by SO

Currently coprocessor assumes control is from a single user

- Future work will look at role-based access control
- Basic SO vs. user separation involves a trivial modification to the cryptlib security kernel

Trusted I/O Path

Standard coprocessor control comes entirely from the host



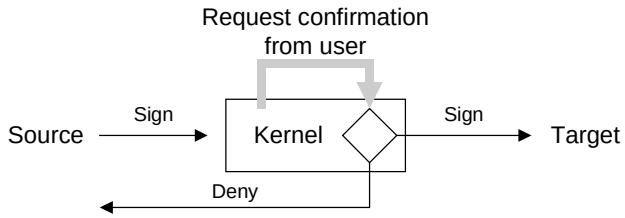
Once the user/SO PIN is entered all bets are off

- Hostile app can request any operation from coprocessor
- Tier 3 is safer than tier 2, much safer than tier 1

Trusted I/O Path

Tier 4 coprocessor can request confirmation of operations from the user

- Implemented as modification to the cryptlib security kernel



- Host requests action
- Kernel requests user confirmation over trusted I/O path
- Lack of confirmation or timeout causes action to fail

Physically Isolated Crypto

Air gap security

- All crypto keys are stored in and processing done on a small satellite orbiting Mars
- Allows use of crypto in countries with GAK laws
  - User in UK, crypto in Ireland or France

Requires a protected session to the coprocessor

- ssh or SSL, preferably with DH keys
- IPsec

Physically Secure Crypto

Coprocessor may need to withstand third-party curiosity

Standard approach

- Embed circuitry in tamper-resistant envelope

Embedded systems are often designed for use in hostile environments

- Use enclosure designed for extreme environments

Image © RTD USA

Example: HiDAN system from Real Time Devices USA

- Heavy-duty aluminium alloy chassis
- Acts as heatsink and provides substantial amount of protection

Physically Secure Crypto (ctd)

Protection level provided

- 85dB EMI shielding from 10-100 MHz
- 80dB EMI shielding to 1 GHz
- Complies with some TEMPEST emission standards
- Build-in power supply module
- Can withstand medium-calibre artillery fire

Image © RTD USA

FIPS 140 level 2 compliant, level 3 compliant if filled with potting mix

Crypto Hardware Acceleration

Conventional crypto

- Coprocessor's onboard CPU can saturate any normal communications channel

Public-key crypto

- FPGAs and ASICs aren't cost-effective on a small scale
- Cheapest crypto accelerator chip: K6-2/450
- AMD and Intel can make it faster cheaper than you can
- Clustered DSPs may offer an advantage
  - Multiple single-cycle multiply-accumulate (MAC) units
  - Low power consumption
  - Glueless multiprocessor support

Availability

Hardware

- Any embedded PC supplier
- Advantech, `http://www.advantech.com/products/sbc.htm`
- Prices from $200 … $much, sometimes < $100 in surplus lots

Software… uhh… ahem…

- Progress stalled since January by thesis
- Exists as demo with hardcoded communications parameters
  - Actual version will support sockets, named pipes, …
- Full version will be released as `http://www.cs.auckland.ac.nz/~pgut001/cryptlib/`
- More information (much more) in my thesis `http://www.cs.auckland.ac.nz/~pgut001/`