ThruGlassXfer

Remote Access, the APT

Key messages for this session

- Current security architecture, flawed (now)
  - Published everything you need to know
    - From first principles, to demonstrations and full code release (PoC) with test framework.
  - The impact will probably be significant
    - No constraints to data theft for remote workers or off-shore partners
- There are no easy answers
  - The paper has some suggestions

Who is this guy, at work?

- Career
  - Blue, create & fix
    - Governance (Technical, Security)
      - Multiple iconic international enterprise organisations
    - Architect / Designer
      - Enterprise perimeters, Data Centre consolidation
    - SysAdmin, Tech Support
  - Red, break & destroy
    - Ethical Hacker / Penetration Tester
      - Pwn'd asia pacific in a business day

Who is this guy, at home?

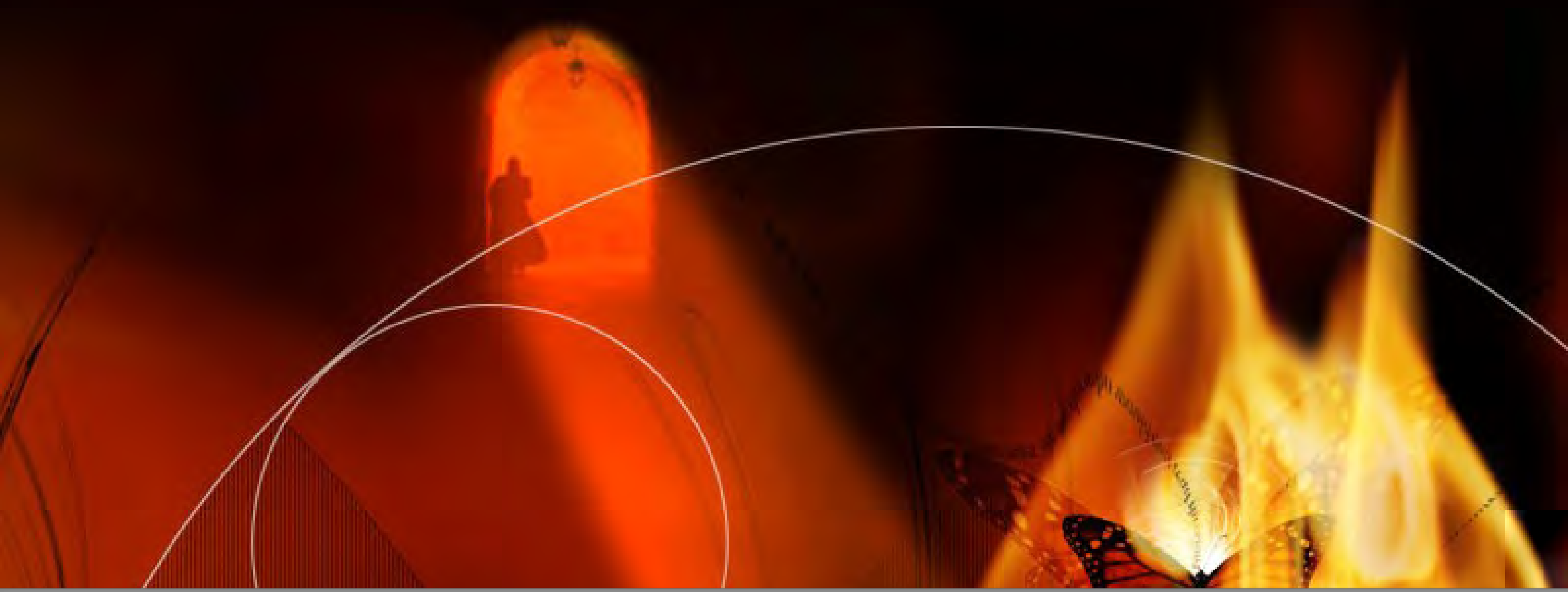
- My time
  - Threat Intelligence
    - “Practical Threat Intelligence” course, BH
    - “Threat Analytics” cloud service
  - OSSTMM
    - “Active Filter Detection” tool
  - Linux distributions
    - “CHAOS” – the super computer for your wallet
    - “Saturn” – scalable distributed storage
  - Barbie car?

Credit to Researchers

- 2013
  - D3adOne (Extracting data with USB HID)
- 2012
  - Ben Toews and Scott Behrens (DLP Circumvention, a Demonstration of Futility)
  - VszA (Leaking data using DIY USB HID device)
- 2011
  - Stephen Nicholas (QuickeR: Using video QR codes to transfer data)
- Also
  - Dfries, Hak5, IronGeek, Mike Szczys, Netragrad, Thomas Cannon
  - And any others who I have not yet found.

PROBLEM SPACE

Framing the Problem

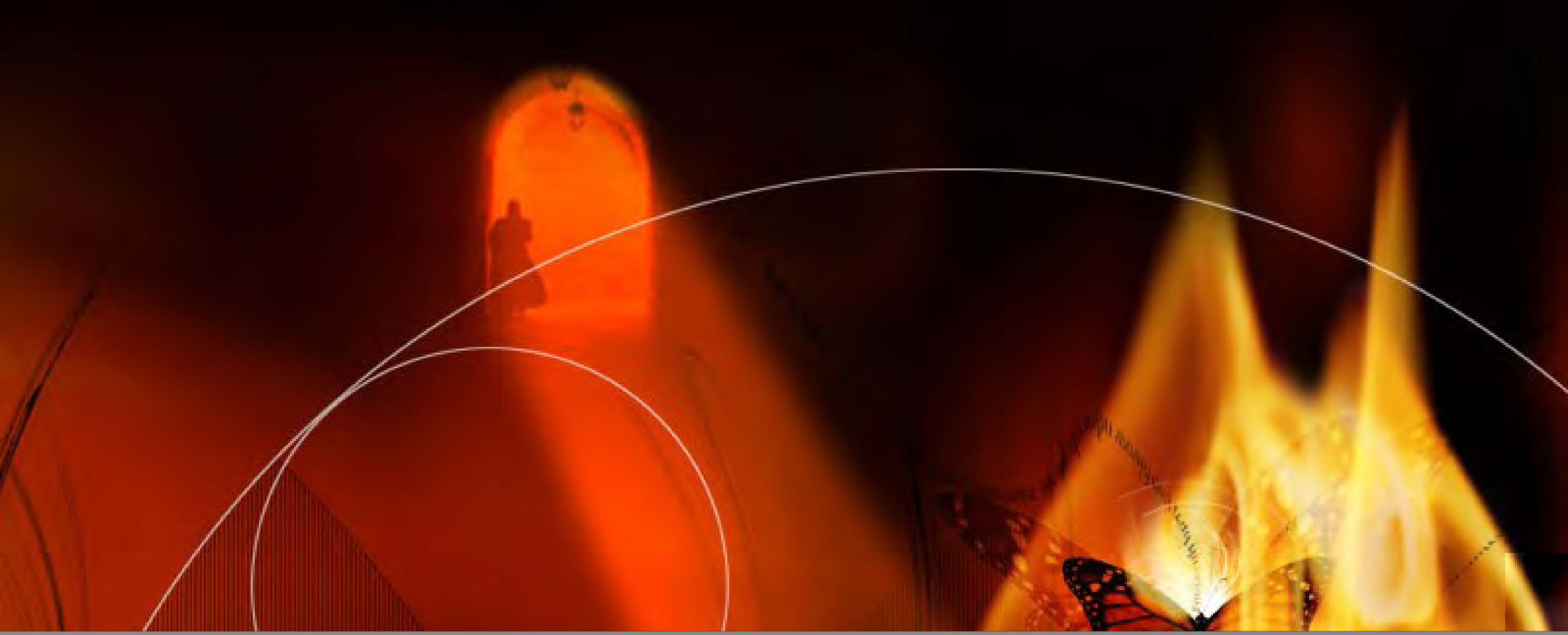


First principles

- Assertion:
  - *Any user controlled bit is a communications channel*
- Validation:
  - The screen transmits large volumes of user controlled bits (imagine the screen as cut fiber optic bundle)
  - Can the screen be transformed into an uncontrolled binary transfer interface?

TECHNOLOGY SOLUTION

Engineering a Proof of Concept



Screen data extraction

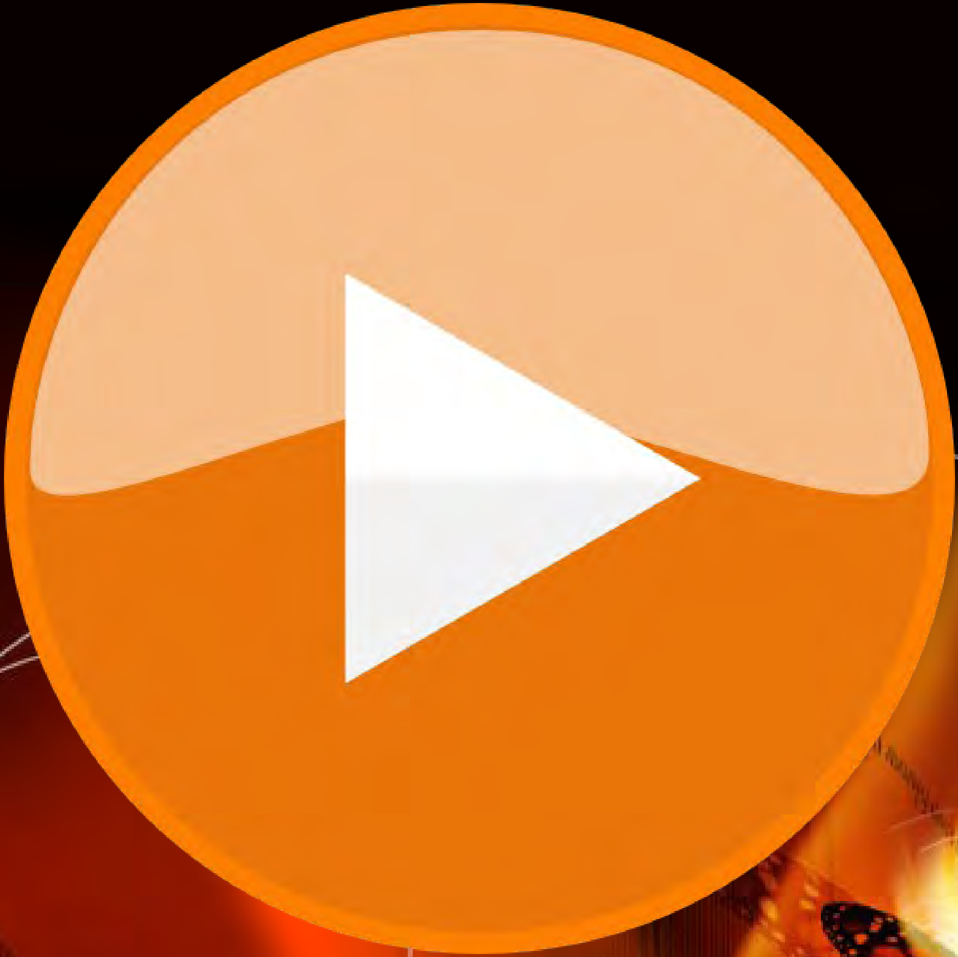
- Terminal Printing (1984)
  - Virtual screen as a multi-use data device
    - DEC VT220 Programmer Reference Manual
  - Ditto for [XYZ]modem protocols
- VHS Tape Backup (1992-1996)
  - Video record/play of compressed binary data
    - Grey-scaled picture of two rows of eight blocks, comprised of more nested blocks
  - ArVid ISA board with AV-in/out (composite)

Real screen data extraction

- Timex DataLink Watch (1994)
  - Address book data transmitted from the screen to a wrist watch
    - Eeprom programmed with light
  - Windows 95 and 98, required CRT
    - Open source (dfries) done with USB TTL LED
  - Transfer rate:
    - 20 seconds to transfer 70 phone numbers

Timex / Microsoft advert

The first computer watch revolution, 1994



Machine recognition

- Quick Response Codes (1994)
  - 1960s: Denso Wave responded to cashiers' need for machine readable Kanji encoded data with 2D barcodes
  - 1990s: Denso Wave wanted to improve performance and did an exhaustive study of printed business materials – QR Code is:
    - Highly distinguished
    - Highly machine recognisable
    - 360 degree scanning

Performance & error correction

- Quick Response Codes (2000-2006)
  - Adopted by the auto industry
  - Formalised as ISO/IEC 18004:2000
    - Rapid scanning capability
    - Automatic re-orientation of the image
    - Inherent error correction
    - Native binary support
  - Revised as ISO/IEC 18004:2006 for model 2
    - Support deformed/distorted codes
    - Capacity up to about 3KB

Optical packet network (L3)

- Zen moment
  - Consider the QR Code as an optical packet captured within the ether of the display device.
  - Datagram network protocol, OSI Layer 3
- Beyond the packet boundary, create a flow
  - Transmitter replaces one code for another
  - Receiver uses video instead of a photo
  - Receiver doesn't exit, just keeps going.

Layer 4 problems

- All new problems:
  - – Unidirectional interface
    - No synchronisation, no signalling, no flow control
    - Requires over-sampling (2-3x)
  - – Oversampling creates duplicates
    - Requires de-duplication
    - Duplicates may be intentional (repeating sequences in the application layer)
- Need for a transport protocol!

Creating transport data flow

- QR code v1 = 14 octets at 15% ECC
  - Take the 1st octet and create "control byte"
  - Create two frames, "Control" and "Data"
- Data Frame
  - Control Byte
    - Bit 0: is always 0 (Data Frame)
    - Bits 1-4: Counter (cycles from 0-15)
    - Bits 5-7: Reserved (unused)
  - Payload of source bytes mod capacity bytes

Creating transport control flow

- Control Frame
  - Control Byte
    - Bit 0: is always 1 (Control Frame)
    - Bits 1-3: Control Type
    - Bits 4-7: Control Sub-Type
  - Payload is control data, as needed
    - File name
    - File size
    - CRC
    - etc

Creating transport control msgs

Control Type	Control Sub-Type	Label	Function
001 (1)	0001 (1)	START/FILENAME	Name of source data
	0010 (2)	START/FILESIZE	Length of source data (octets)
	0011 (3)	START/QR_VER	QR code version
	0100 (4)	START/QR_FPS	QR code frames per second
	0101 (5)	START/QR_BYTES	QR code octets per frame
010 (2)	0001 (1)	STOP/PAUSE	Transmission paused
	0010 (2)	STOP/COMPLETE	Transmission completed
	0011 (3)	STOP/CANCEL	Transmission cancelled
011 (3)	0001 (1)	STATUS/SINCE	Status since last status

TGXf Transport Protocol +

- One way data transfer between two or more peers
  - Features (at Layer 4-7)
    - Supports high latency
    - Supports interrupted transfers
    - Includes error detection
  - Requires (of Layer 3)
    - Either 1, 2, 5, 8 or 10 Frames Per Second (FPS)
    - QR Code version 1, 2, 8 or 15
    - Binary encoding, Type M (15%) error correction

TGXf Layer 3 Configurations

- Supported QR code versions
  - No real impact on Layer 4 (MTU)

Version	Mode	ECC	Frame Capacity	Reliable Capacity
1	Binary	M (15%)	14 bytes per frame	10 bytes per frame
2	Binary	M (15%)	26 bytes per frame	22 bytes per frame
8	Binary	M (15%)	152 bytes per frame	148 bytes per frame
15	Binary	M (15%)	412 bytes per frame	408 bytes per frame

  - ECC is dynamic and can exceed the binary payload capacity, resulting in a frame of a different version (automatically increases resolution)

TGXf Hello World – 1/1:1

- Control Frame
  - Control Byte
    - Bit 0: Control (1)
    - Bits1-3: START (1)
    - Bits4-7:
      FILENAME (1)
  - Payload
    - “helloworl”
  - Encode as QR code version 8 datagram

TGXf Hello World – 1/1:2

- Control Frame
  - Control Byte
    - Bit 0: Control (1)
    - Bits1-3: START (1)
    - Bits4-7:
      FILESIZE (2)
  - Payload
    - 13 octets
  - Encode as QR code version 8 datagram

TGXf Hello World – 1/1:5

- Control Frame
  - Control Byte
    - Bit 0: Control (1)
    - Bits1-3: START (1)
    - Bits4-7: QRCODE_BYTES (5)
  - Payload
    - 148 octets
  - Encode as QR code version 8 datagram

TGXf Hello World – 1/1:4

- Control Frame
  - Control Byte
    - Bit 0: Control (1)
    - Bits1-3: START (1)
    - Bits4-7:
      QRCODE_FPS (4)
  - Payload
    - 5 fps
  - Encode as QR code version 8 datagram

TGXf Hello World – 0/data

- Data Frame
  - Control Byte
    - Bit 0: Data (0)
    - Bits1-4: Counter (0)
  - Payload
    - “Hello World!”
  - Encode as QR code version 8 datagram

TGXf Hello World – 1/2:2

- Control Frame
  - Control Byte
    - Bit 0: Control (1)
    - Bits1-3: STOP (2)
    - Bits4-7:
      COMPLETE (2)
  - Payload
    - CRC32
  - Encode as QR code version 8 datagram

TGXf Result – visual modem

If you see this in Smart Auditor, you've lost

TGXf Data Rates

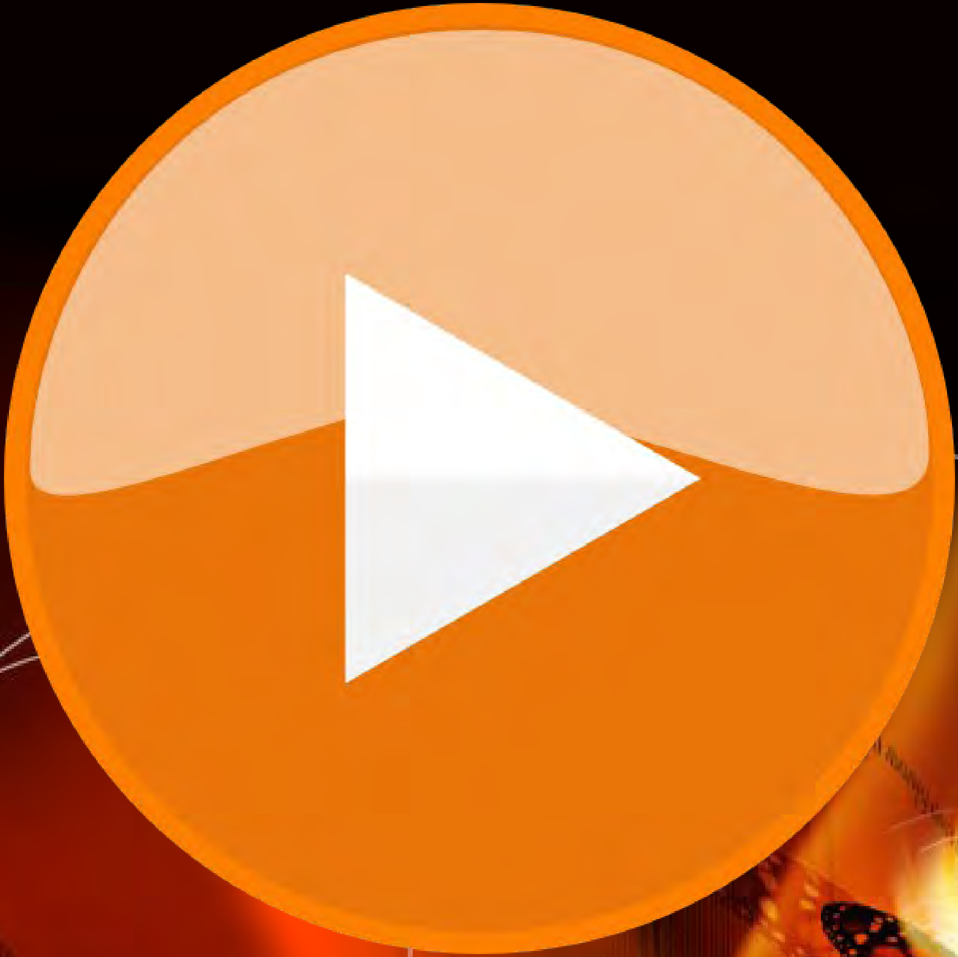
- Recall the supported QR Code versions
  - Updating our Layer 3 configurations table with FPS values, we get the following.

Version	Reliable Capacity	FPS (1 -> 10) x 8 bits
1	10 bytes per frame	80bps -> 800 bps
2	22 bytes per frame	176 bps -> 1,760 bps
8	148 bytes per frame	1,184 bps -> 11,840 bps
15	408 bytes per frame	3,264 bps -> 32,640 bps

  - I.e. 80 bps to 32 kbps
  - Arbitrarily limited only by the receiver

TGXf a PDF from Youtube

YouTube is the new DropBox



Another version

- Recall the supported QR Code versions
  - Updating our Layer 3 configurations table with resolutions, we get the following.

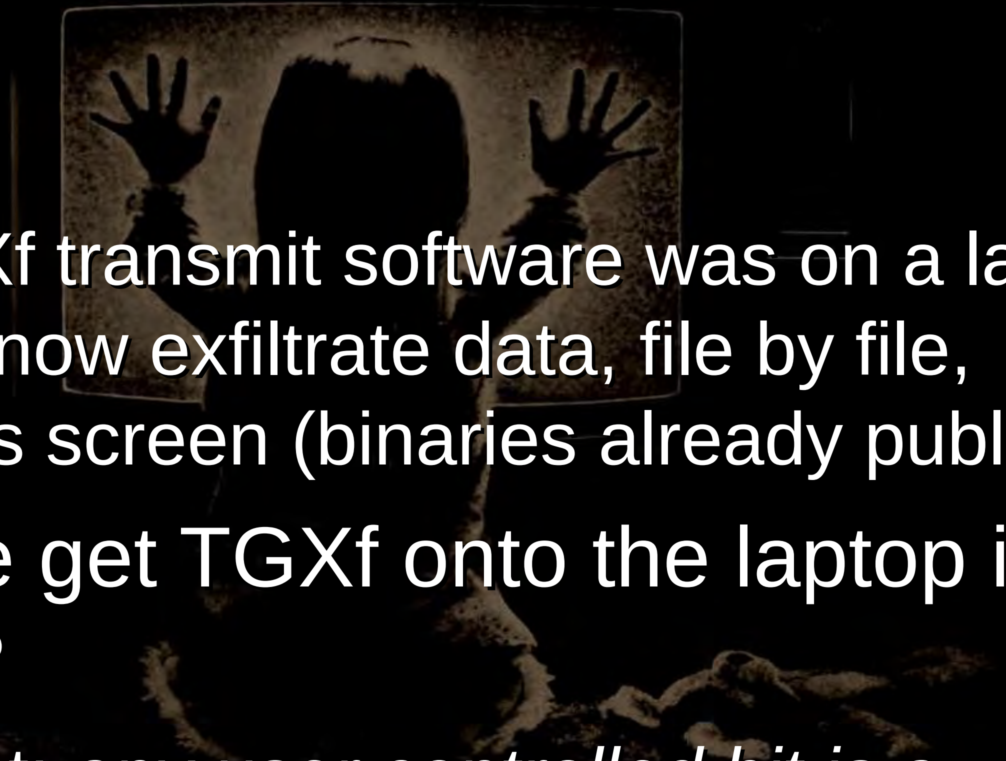
Version	Reliable Capacity	Resolution
1	10 bytes per frame	21 x 21 pixels
2	22 bytes per frame	25 x 25 pixels
8	148 bytes per frame	49 x 49 pixels
15	408 bytes per frame	77 x 77 pixels

- Previous examples scaled the code
  - Lets look at a native version 1 example ..

TGXf a PDF from BASH

ANSI generated QR codes pass through SSH jump hosts

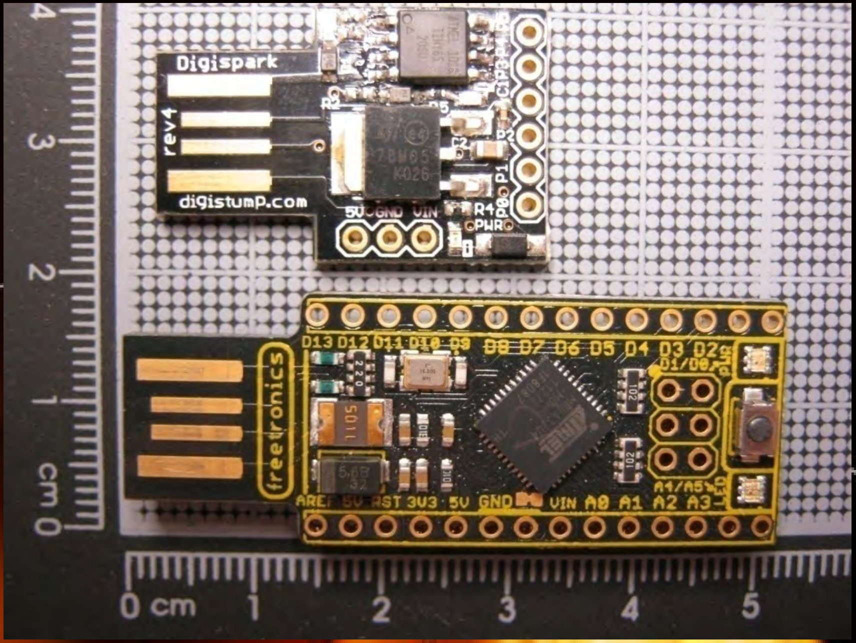
Technology check-point (1/3)



- So!
  - If the TGXf transmit software was on a laptop we could now exfiltrate data, file by file, through its screen (binaries already public)
- How do we get TGXf onto the laptop in the first place?
  - Recall that: *any user controlled bit is a communications channel* ..
  - And .. we have a keyboard!

Digital Programmable Keyboard

- Arduino Leonardo
  - USB HID Keyboard
    - No drivers needed!
    - Keyboard.println("x")
  - Open source platform
    - Heaps of support!
- Digispark (top)
  - 6KB of flash
- Leostick
  - 32KB of flash



What to type?

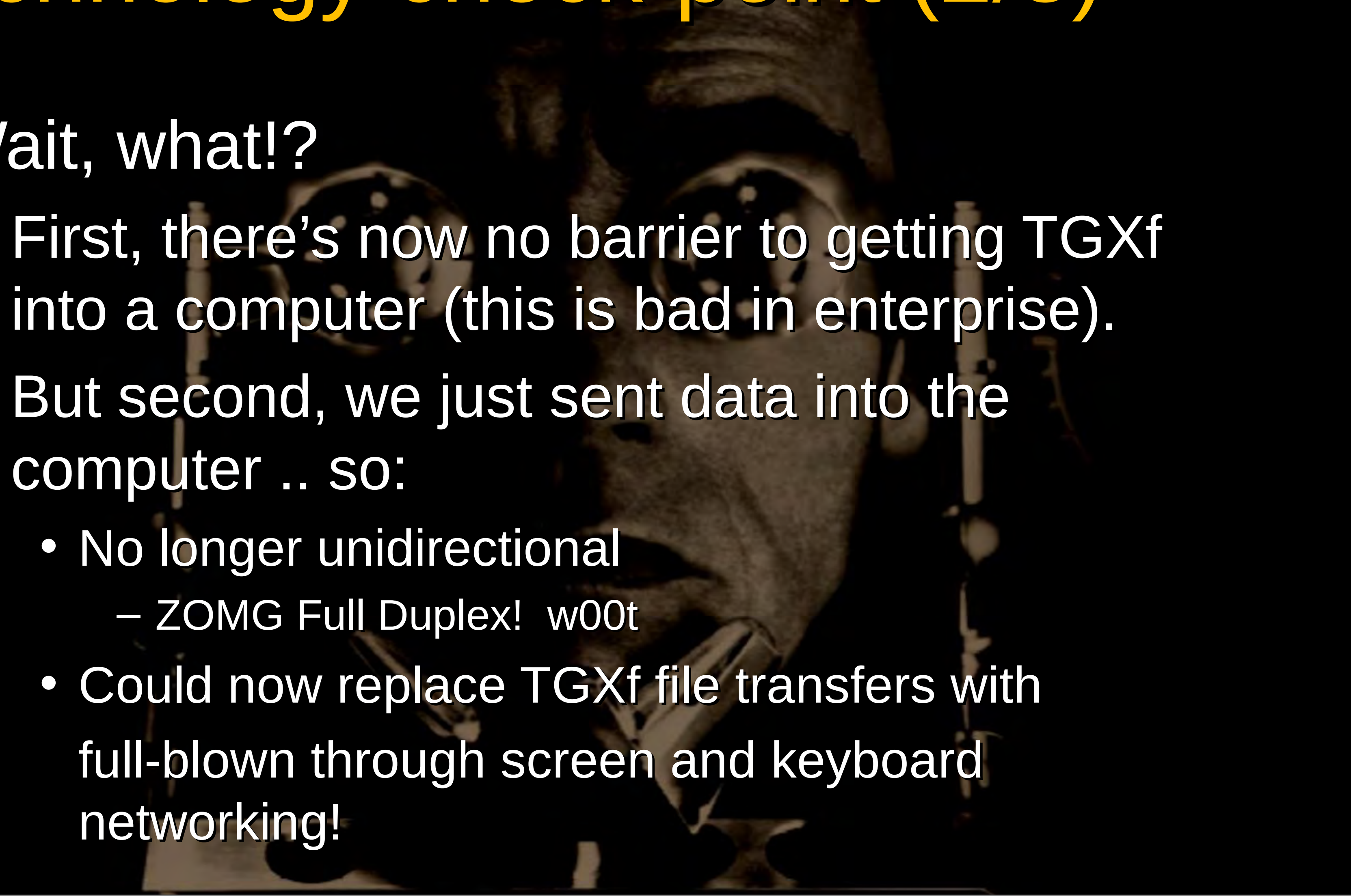
- Source code (text) would be easy to send but then needs to be compiled .. meh
- Send statically compiled binary
  - Gzip TGXf transmit binary (~80->25KB)
  - Hexdump the .gz (byte = 2 chars; 0-9, a-f)
- Receive via text editor
  - “Type” it in, structured
    - Bash (printf) or Perl (print)
  - Save, chmod and run script, gunzip result!

Typing a BASH2BIN script

Uploading to Lin64 via Win32 over SSH via keyboard

Technology check-point (2/3)

- Wait, what!?
  - First, there's now no barrier to getting TGXf into a computer (this is bad in enterprise).
  - But second, we just sent data into the computer .. so:
    - No longer unidirectional
      - ZOMG Full Duplex! w00t
    - Could now replace TGXf file transfers with full-blown through screen and keyboard networking!



Keyboard Interface

- USB HID Keyboard interface
  - Polled interface, each 1ms
    - Typical implementations send one “key” packet followed by one “null” packet (key clear)
    - Not necessary, but still implemented
  - Contains up to 6 keyboard keys (by code)
    - Note – no native binary mode
  - Automatically de-duped (no one key twice)
    - Note – data removed irretrievably

TKXf – Keyboard Transport

- Same as TGXf – USB HID packet is L3
  - Still unidirectional
    - Though status LEDs could be used
  - Create binary payload by encoding data in hexadecimal
    - Halves throughput: 3 octets/pkt/ms
    - Retained “key clear” packet: 3 octets/pkt/2ms
  - Correct for de-duplication by creating a de-dupe layer that re-dupes at the receiving end
    - Simple positional reference based encoding

TKXf – Transport Protocol

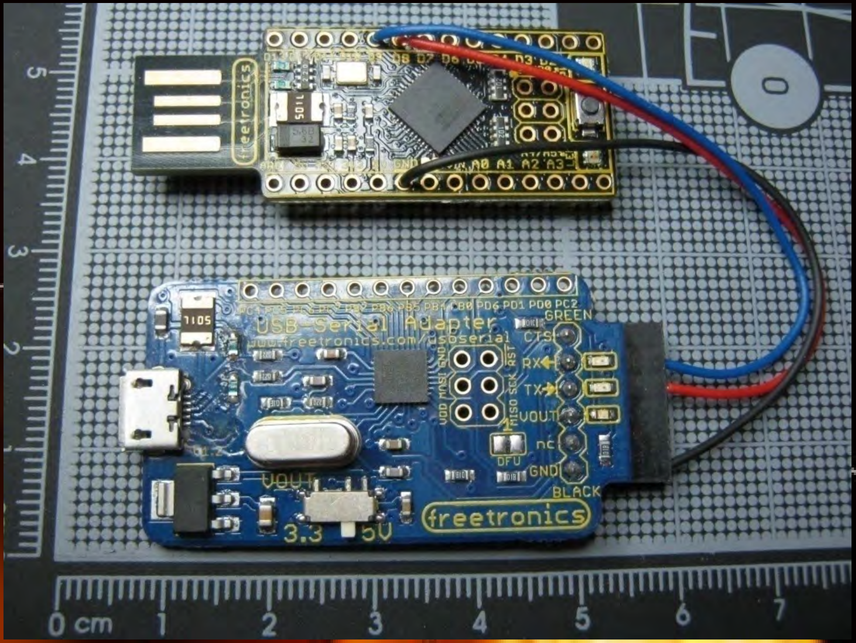
- 6 char packets are too small for a control header
  - Bookended "sequence" instead of "packet"

• Data	= "space"	= 0x2C/0x20
• Control/Start	= "comma"	= 0x36/0x2C
• Control/Stop	= "period"	= 0x37/0x2E

  - Process as a "stream"
- And let's ignore "file" based transfers ..

TKXf – "Keyboard Stuffer"

- Target Arduino (top)
  - USB HID Keyboard
    - Encodes received raw/binary data as keys
  - Alter "Keyboard" library to expose HID packet (12x faster ++)
- Attacker Arduino
  - USB Serial Interface
    - Sends raw/binary octets to Target Arduino

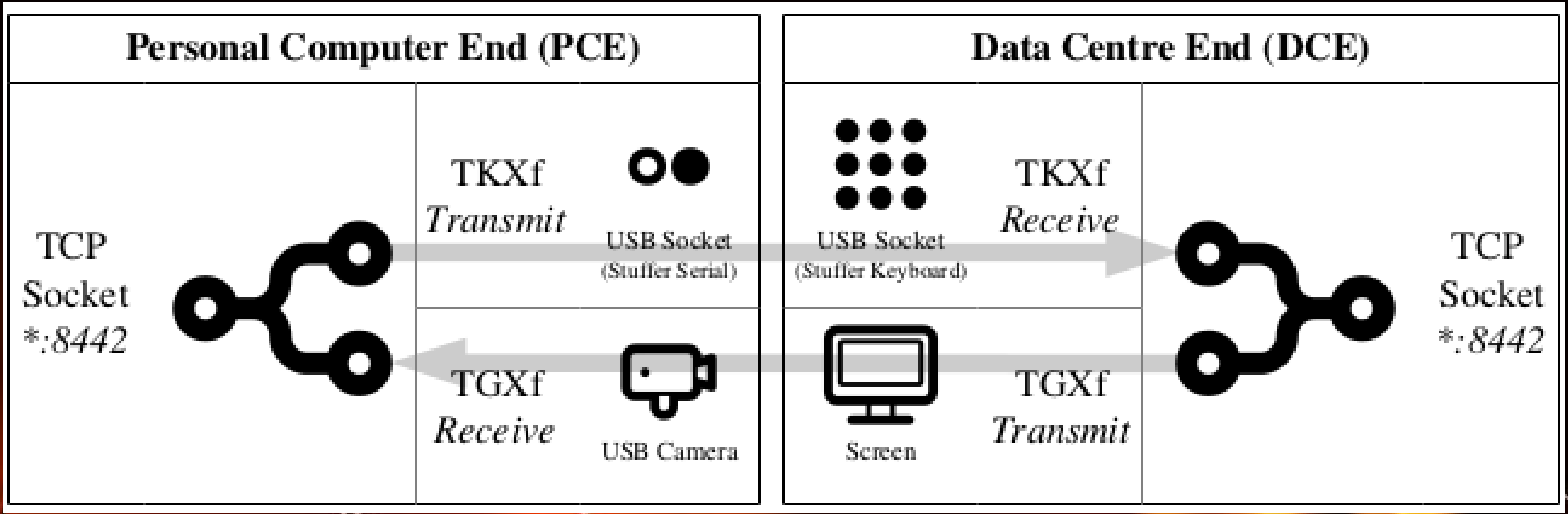


TGXf note

- One note on TGXf before we integrate TGXf and TKXf
  - If we remove the control frames (layer) from TGXf it is capable of “streams” rather than “files”

- Now we can assemble the *Through Console Transfer* application!

TCXf Application Architecture



Technology check-point (3/3)

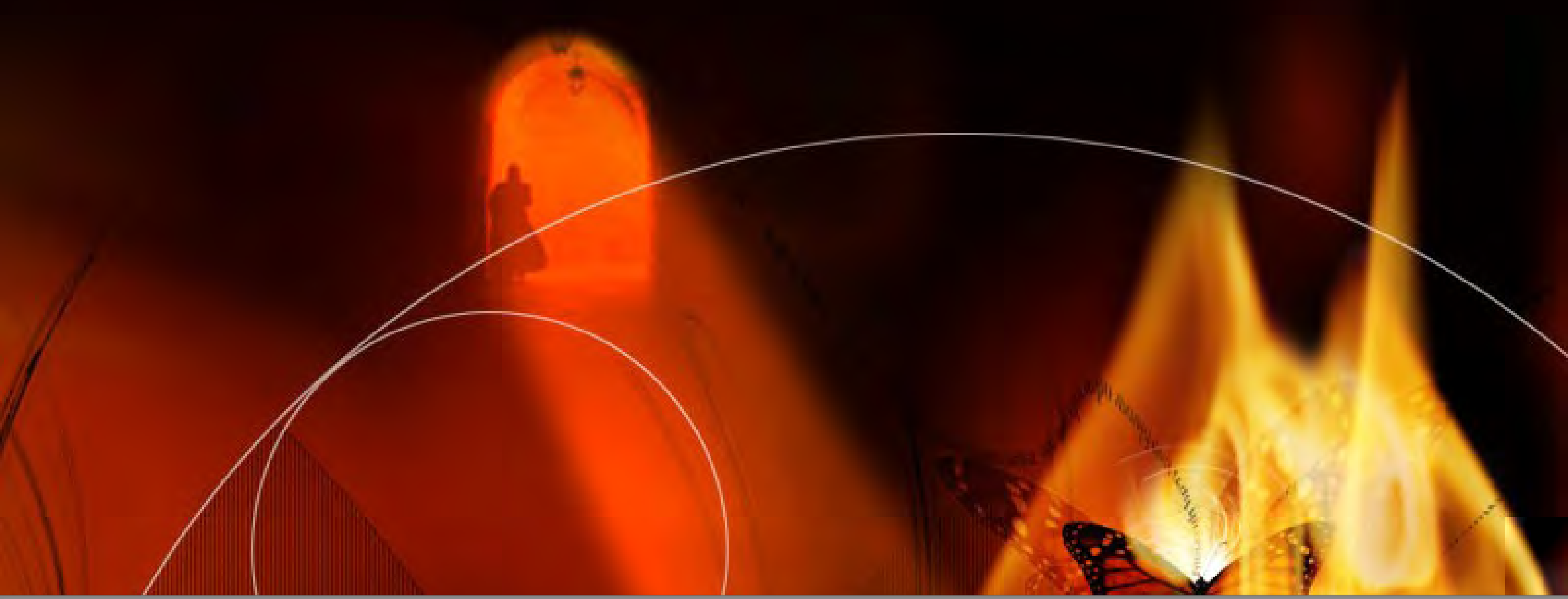
- TCXf
  - TKXf reference impl. has 12kbps max, up
    - Could probably get this up to 32kbps
      - Use *Key clear* packet with second character set (x2)
      - Use base64 for 4 bytes per 3 hex values (+1/3)
  - TGXf reference impl. has 32kbps max, down
  - Features
    - Bi-directional, binary clear, serial connection
    - Native network socket interface
    - Insane portability / Massive vulnerability

TCXf IP Network Evolution

- PPP over the Screen and Keyboard
  - On the target device;
    - sudo pppd 10.1.1.1:10.1.1.2 debug noccp nodetatch pty “netcat localhost 8442”
      - Note the privilege required to create a NIC (We already had a full-duplex socket without it)
  - On the attackers device;
    - sleep 2; sudo pppd noipdefault debug noccp nodetatch pty “netcat localhost 8442”

ARCHITECTURE

POC Impact on the Enterprise Architecture



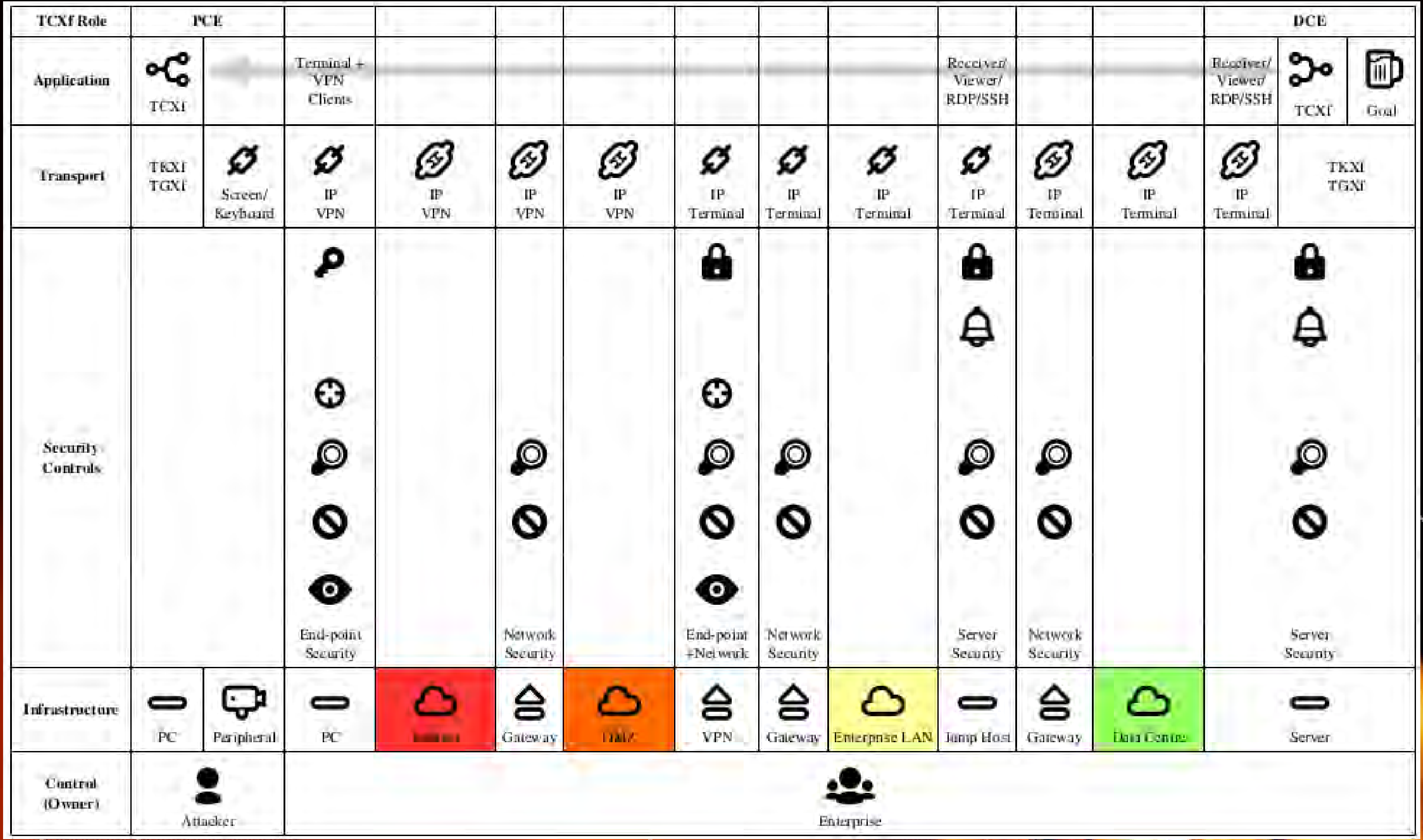
ESA Context?

- Time to be Enterprise Security Architects
  - Firstly, what are TGXf, TKXf and TCXf?
    - In the vulnerability taxonomy we are dealing with as “storage based” *covert channel* attacks
  - Secondly, where’s the enterprise?
    - So far we’ve been working from a local computer context
    - But in enterprise we abstract the screen and keyboard (on the organisation’s side) ..

This is enterprise @ L7

- Remote access
  - VMware
  - Citrix
  - RDP
  - VNC
  - SSH
  - etc ad nausea
- Console abstraction

TCXf Enterprise Impact



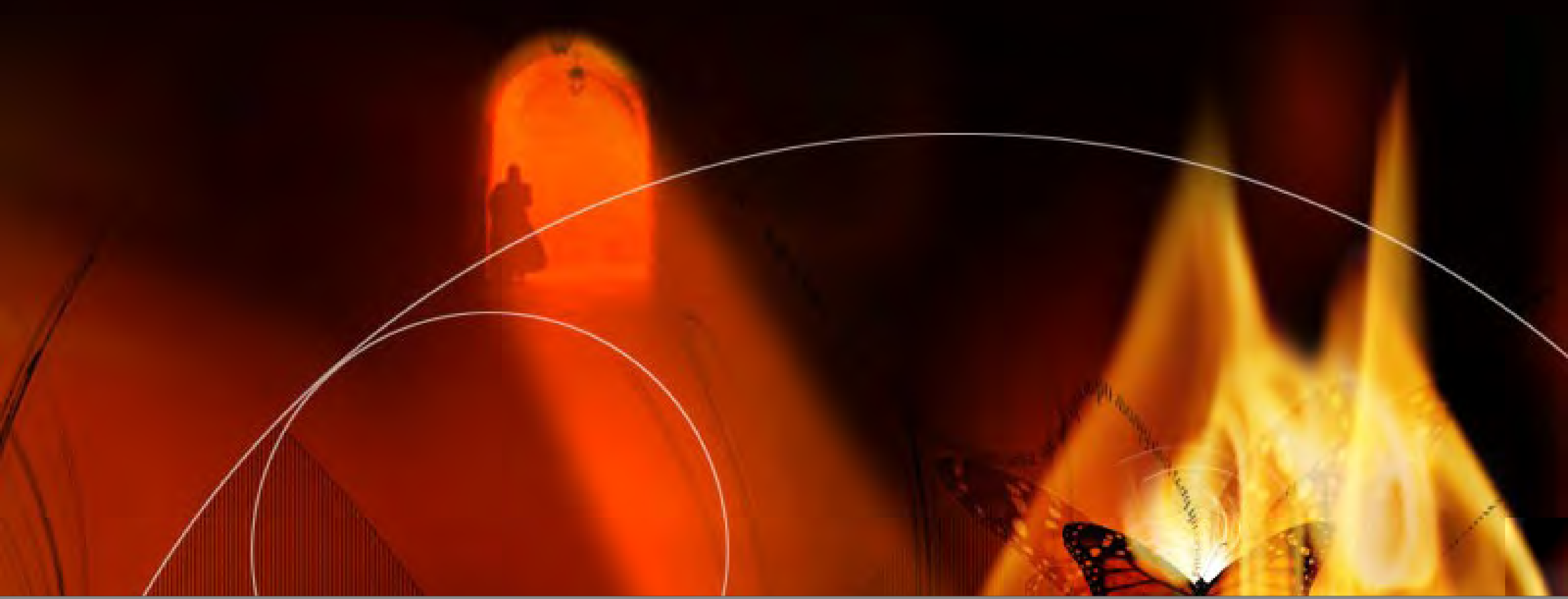
TCXf Role	PCE												DCE		
Application	TCXf		Terminal + VPN Clients							Receiver/ Viewer/ RDP/SSH			Receiver/ Viewer/ RDP/SSH	TCXf	Goal
Transport	TKXf TGXf	Screen/ Keyboard	IP VPN	IP VPN	IP VPN	IP VPN	IP Terminal	IP Terminal	IP Terminal	IP Terminal	IP Terminal	IP Terminal	IP Terminal	TKXf TGXf	
Security Controls			End-point Security		Network Security		End-point +Network	Network Security		Server Security	Network Security		Server Security		
Infrastructure	PC	Peripheral	PC	Internet	Gateway	DMZ	VPN	Gateway	Enterprise LAN	Jump Host	Gateway	Data Centre	Server		
Control (Owner)	Attacker		Enterprise												

TCXf PPP via XPe Thin Client

Attacker Laptop → Corp XPe Thin Client → Corp Linux App Server

TECHNOLOGY SOLUTION 2

Engineering a better Proof of Concept



clientlessTGXf (New, Xmas '14!)

- Making good on the ASCII threat
  - TGXf is a Transport Protocol at Layer 4
  - Datagram Protocols at Layer 3 could be;
    - Graphic: Pixels, Images (i.e. F500 logos)
    - Text: Letters, Words, Phrases
  - clientlessTGXf PoC (published December)
    - Use text letters to enable clientless
    - Minimal server side IOCs
    - Demos futility of QR code detection

clientlessTGXf from BASH

- POC was tested on this terminal setup;
  - – Monospace Bold font at 16pt, black text on white BG
- About 300 bytes of script (counter+data, FD3)

```
IFS="" ; LANG=C ; c=0 ; while read -s -u 3 -d '' -r -n 1
i ; do  printf -v b "%i" "'$i" ;  echo " $((($c&128)>>7))$
((($c&64)>>6))$((($c&32)>>5))$((($c&16)>>4))$
((($c&8)>>3))$((($c&4)>>2))$((($c&2)>>1))$(($c&1))  $
((($b&128)>>7))$((($b&64)>>6))$((($b&32)>>5))$
((($b&16)>>4))$((($b&8)>>3))$((($b&4)>>2))$
((($b&2)>>1))$(($b&1))" ;  ((c++)) ;  (($c==256)) &&
c=0 ;  sleep 0.005 ; done
```

clientlessTGXf & HDMI Capture

- Capture (1kbps)
  - HDMI MITM
    - AverMedia Game Capture II (with remote!)
    - Up to 1920x1080x30fps captured to MP4 on USB Mass Storage
- Recovery (100bps)
  - Video Processing
    - Data recovered from MP4 file by Linux application

clientlessTGXf in the Red Room

- USB HDD and Red Room Protocols
  - A device can enter
    - Must be blanked (except for firmware)
    - Shouldn't have a problem getting capture equipment into the Red Room
  - A device can leave
    - Must be blanked (except for firmware)
    - Will have a problem bringing back the USB mass storage with MP4 ..?

Hiding USB keys?

Captain Koons says "be creative" ..

clientlessTGXf upload

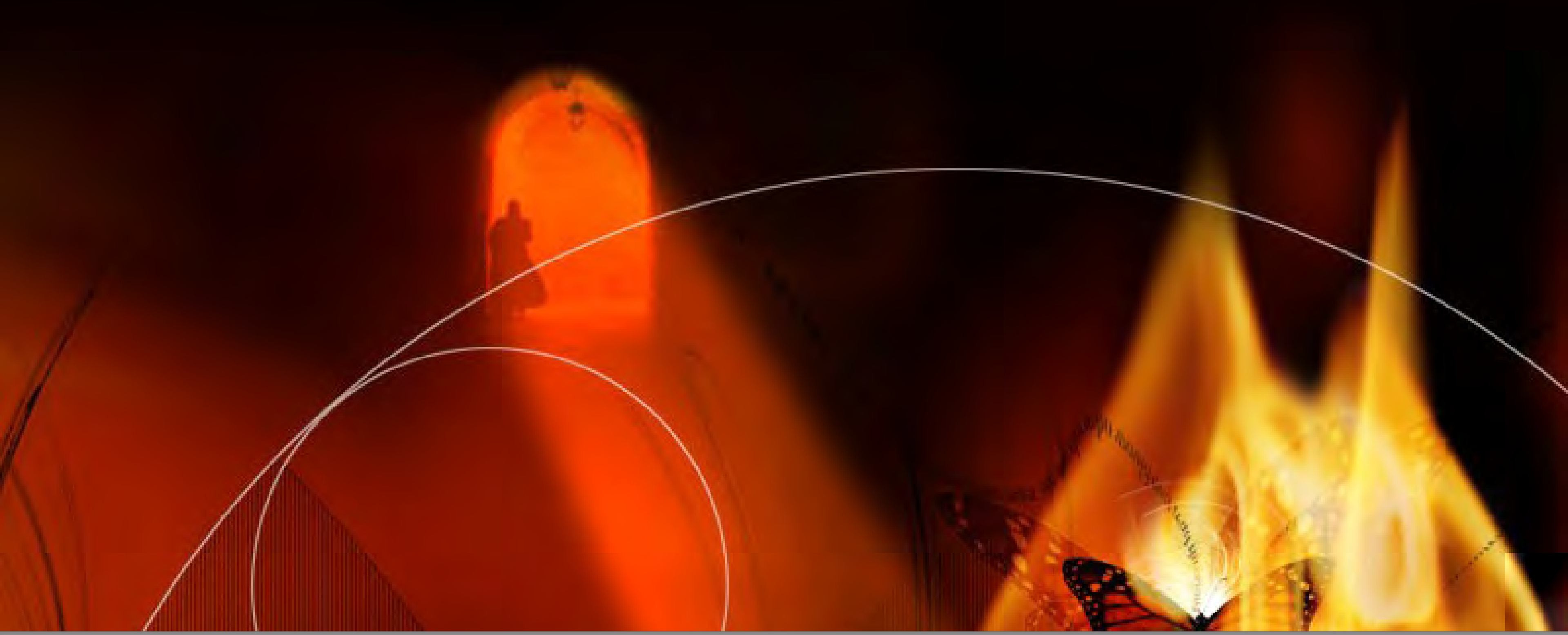
Native 1920x1080 HDMI capture with clientlessTGXf BASH upload

clientlessTGXf download

Decoding TGXf video stream to disk as original file

TECHNOLOGY SOLUTION 3

Engineering a faster Proof of Concept



highspeedTGXf (New, Today!)

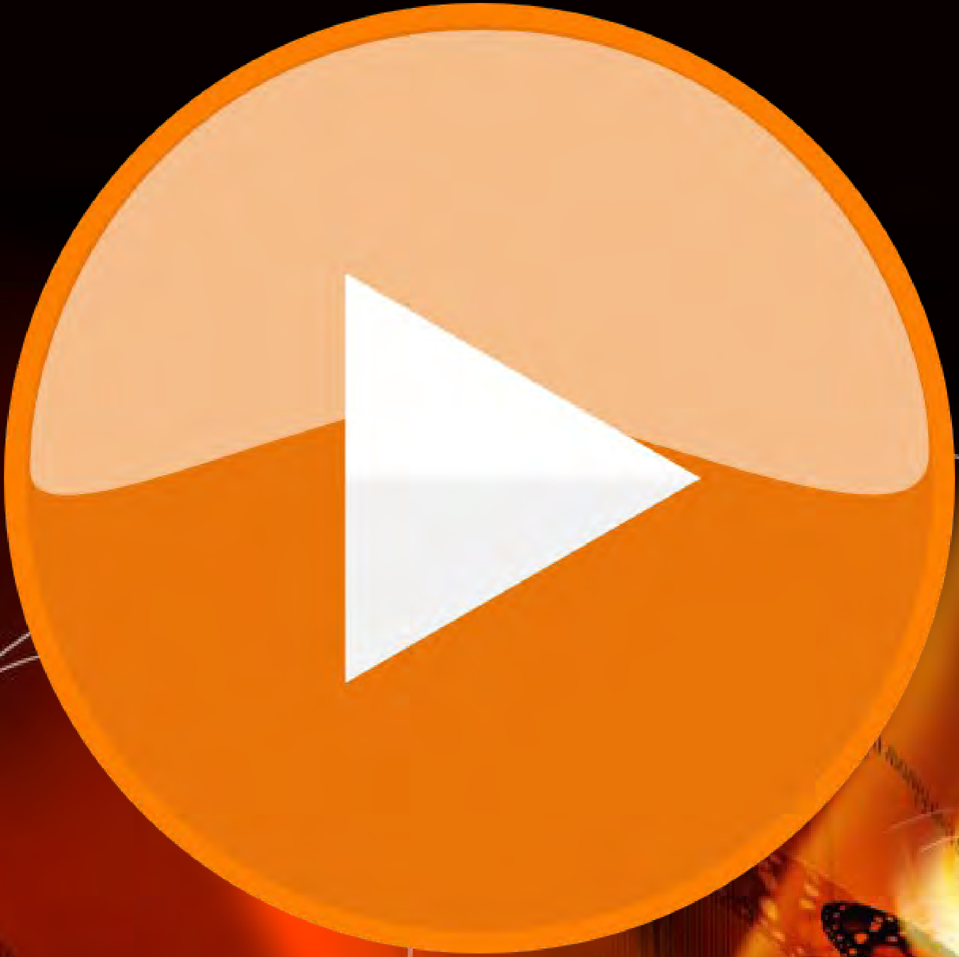
- Making good on the PIXEL threat
  - TGXf updated Transport Protocol at Layer 4
    - CRC32 per Frame/Packet
  - Datagram Protocol at Layer 3 is pixels;
    - HTML5 canvas and Javascript
  - hsTGXf PoC (published today)
    - Use ~20k of Javascript to enable clientless
    - Again, minimal server side IOCs
    - Again, demos futility of targeting a specific implementation

hsTGXf & HDMI Capture

- Capture (1.3Mbps @ 2FPS, 1BPP)
  - HDMI MITM
    - AverMedia Game Capture II (with remote!)
    - Up to 1280x720x60fps captured to MP4 on USB Mass Storage
    - Demo 1240x650x2fps (100,750 bytes per frame)
- Recovery (per before)
  - Video Processing
    - Data recovered from MP4 file by Linux application

highspeedTGXf “client”

Uploading the HTML5 and Javascript client via keyboard



highspeedTGXf upload

Native 1280x720 HDMI capture with web browser upload

hsTGXf download

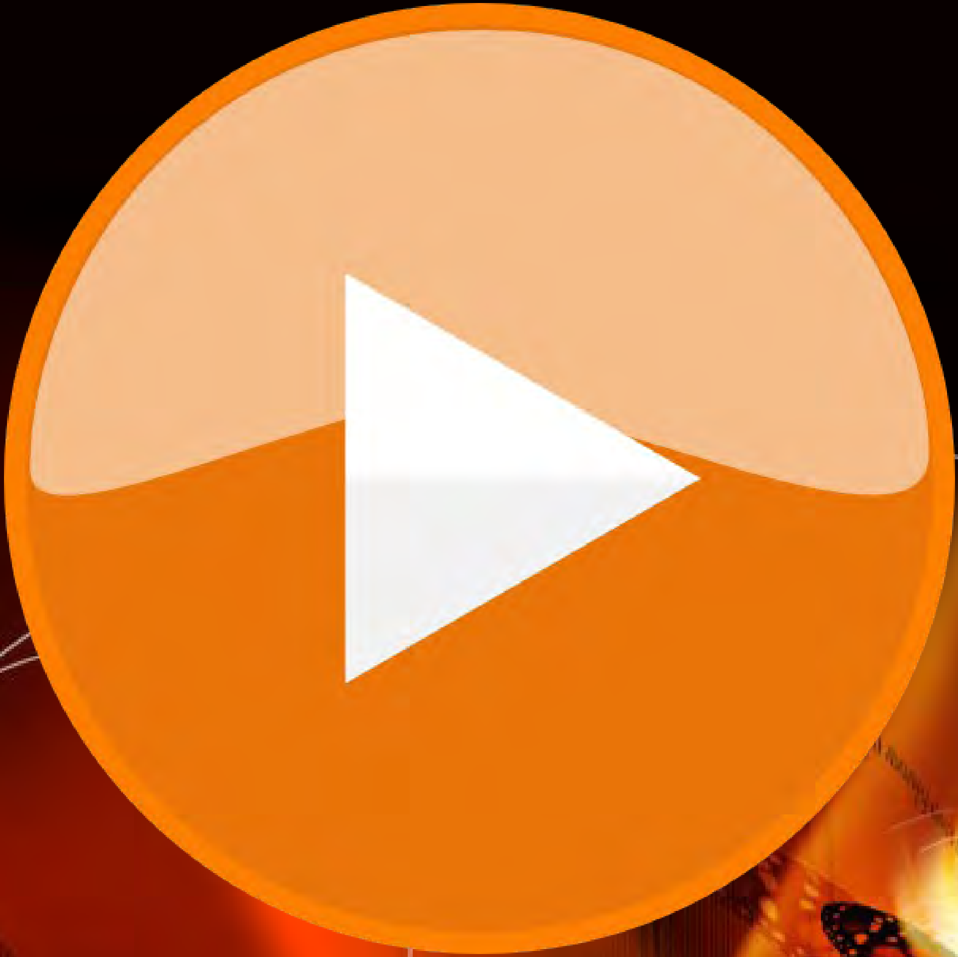
Decoding 1.3Mbps TGXf video stream to disk as original file

hsTGXf & HDMI Capture

- Capture (4.7Mbps @ 8FPS, 1BPP)
  - HDMI MITM
    - Black Magic Design DeckLink Mini Recorder
    - Up to 1280x720x60fps captured to local SATA HDD
    - Demo 1240x650x8fps (100,750 bytes per frame)
- Recovery (per before)
  - Video Processing
    - Data recovered from AVI file (YUV) by Linux application

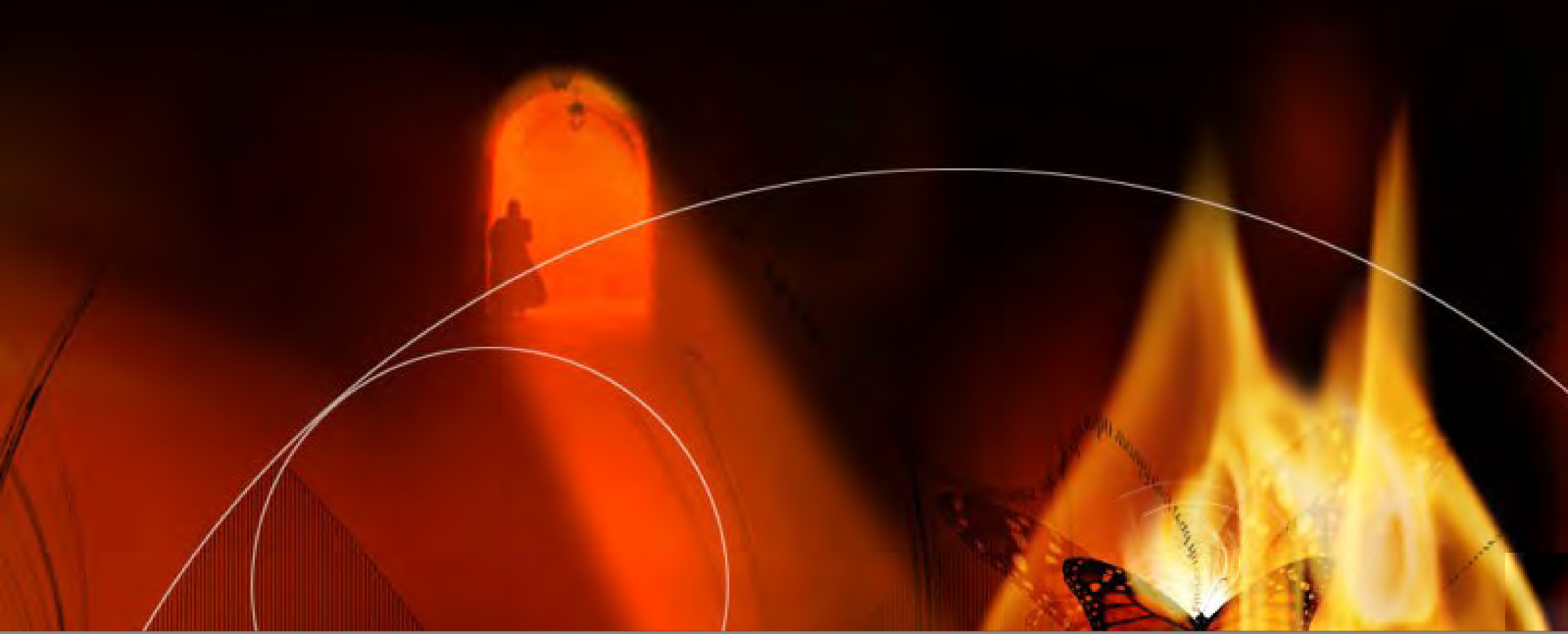
hsTGXf download

Decoding 4.7Mbps TGXf video stream to disk as original file



Architecture

Back to the impact on Enterprise Architecture



Architectural Analysis

- Human versus Machine?
  - Leaving out the PPP example, no variation in access was granted to the user.
    - TCXf can only type and read what the user can.
  - The distinct properties of delta seem to be;
    - **Volume**: transfer rate in bits per second or number of bits at rest, and;
    - **Accuracy**: of data transferred or stored, and;
    - **Structure**: of the data transferred, and;
    - **Utility**: of the over-arching capability.

The Problem? Legal ..

- Use -v- Disclosure, Off-shore (Australian Privacy Act)
  - An Australian organisation (APP entity) that holds personal information [6.6], and makes that personal information available for use [6.8] but not disclosure [6.11(b)] to an overseas recipient [8.5] is accountable [8.1] for ensuring that the overseas recipient does not breach the Australian Privacy Principles [8.2]. <u>Personal information is disclosed when the Australian organisation releases the subsequent handling of the information from its effective control [8.8], which includes disclosure through unauthorised access, if it did not take reasonable steps to protect the personal information from unauthorised access [8.10].</u> The reasonable steps include implementing strategies to manage, amongst other things, ICT Security, data breaches and regular monitoring and review [11.8].
- But what's a reasonable monitoring step?

History – Lampson, 1973

- “A note on the Confinement Problem”
  - *Enforcement: The supervisor must ensure that a confined program's input to covert channels conforms to the caller's specifications. This may require slowing the program down, generating spurious disk references, or whatever, but it is conceptually straightforward. The cost of enforcement may be high. <u>A cheaper alternative (if the customer is willing to accept some amount of leakage) is to bound the capacity of the covert channels.</u>*

History – DoD 1983-2002 (1/4)

- "Trusted Computer Security Evaluation Criteria (TCSEC)"
  - Covert storage channels and Covert timing channels in B2 and B3
    - Content comes almost directly from Lampson
    - Two key points – performance and the acceptable thresh-holds ("not permissible" versus "auditable")

History – DoD 1983-2002 (2/4)

- Performance
  - *A covert channel bandwidth that exceeds a rate of one hundred (100) bits per second is considered "high" because 100 bits per second is the approximate rate at which many computer terminals are run. It does not seem appropriate to call a computer system "secure" if information can be compromised at a rate equal to the normal output rate of some commonly used device.*
  - Take note!
    - *TGXf v1f1 = 80bps; TGXf v1f2 = 160bps;*
      BASH TGXf example = v1f5
    - HDMI = 1080 x 1920 x 24bit x 24fps
      = 150MBps (> 1Gbps)

History – DoD 1983-2002 (3/4)

- Acceptability
    - *In any multilevel computer system there are a number of relatively low-bandwidth covert channels whose existence is deeply ingrained in the system design. Faced with the large potential cost of reducing the bandwidths of such covert channels, it is felt that those with maximum bandwidths of less than one (1) bit per second are acceptable in most application environments. Though maintaining acceptable performance in some systems may make it impractical to eliminate all covert channels with bandwidths of 1 or more bits per second, it is possible to audit their use without adversely affecting systems performance.*

History – DoD 1983-2002 (4/4)

- Acceptability (cont)
  - *This audit capability provides the system administration with a means of detecting – and procedurally correcting – significant compromise. Therefore, a Trusted Computing Base should provide, wherever possible, the <u>capability to audit the use of covert channel mechanisms with bandwidths that may exceed a rate of one (1) bit in ten (10) seconds.</u>*

Punch-line – Use and Off-shore

- The new equilibrium ..
  - Use –v– Disclosure (HIPAA, FISMA, etc)
    - Are functionally identical (Display = Upload)
    - There is no pragmatic difference
  - Off-shoring / Right-Sourcing / Best-shoring
    - .. as the short-form for “remote access for un/semi-trusted users to access sensitive data on shore” ..
    - .. if you like your data to be yours alone ..
    - Is not currently, and is unlikely to ever be, “safe”
  - How many bps data loss is "too many" to accept?

Thank-you!

– Thanks to DefCon

– Thanks to my wife and daughter

– ThruGlassXfer

– Information site:	http://thruglassxfer.com/
– Project site:	http://midnightcode.org/projects/TGXf/
– Contact me:	Ian.Latter@midnightcode.org

(If you're talking to me on social media, it's not me)