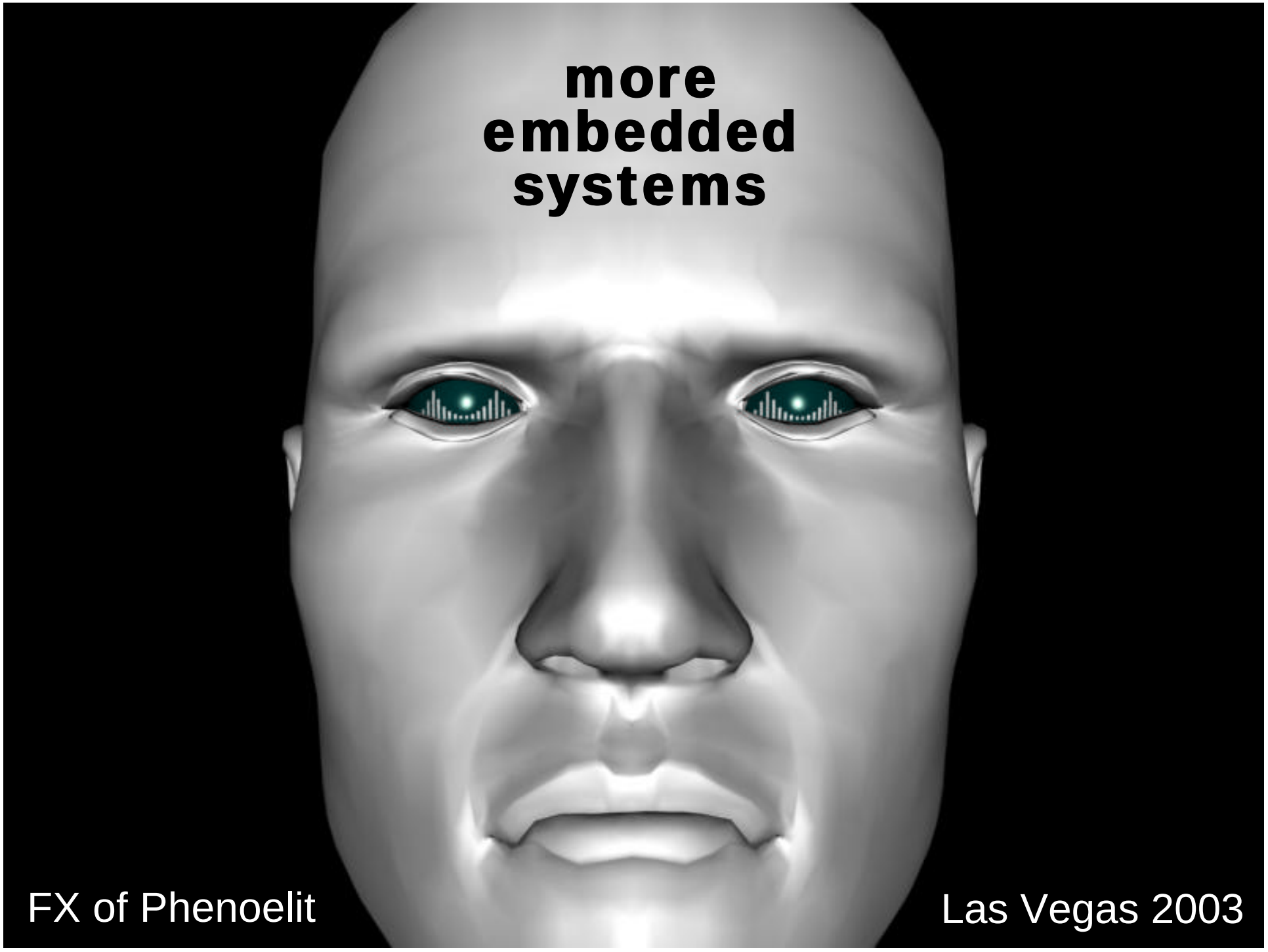

more
embedded
systems
FX of Phenoelit
Las Vegas 2003

Agenda

- Hacking the Matrix
- GSM 3G Basics
- GPRS backbone hacks
- Anonymous HTTP via WAP
- Siemens S55 Vulnerabilities
- Phenoelit‘s usual Cisco Øday vulnerability and exploit

Hacking the MATRIX

- Enterasys Matrix E1
- Vulnerabilities in the Matrix:
  - SSH can only fork 10 times
  - unfinished connections stay open
  - TCP ISN 64k rule on switch ports
  - OSPF neighbors added to neighbor list in state HELO
  - HTTP Server negative content-length integer bug

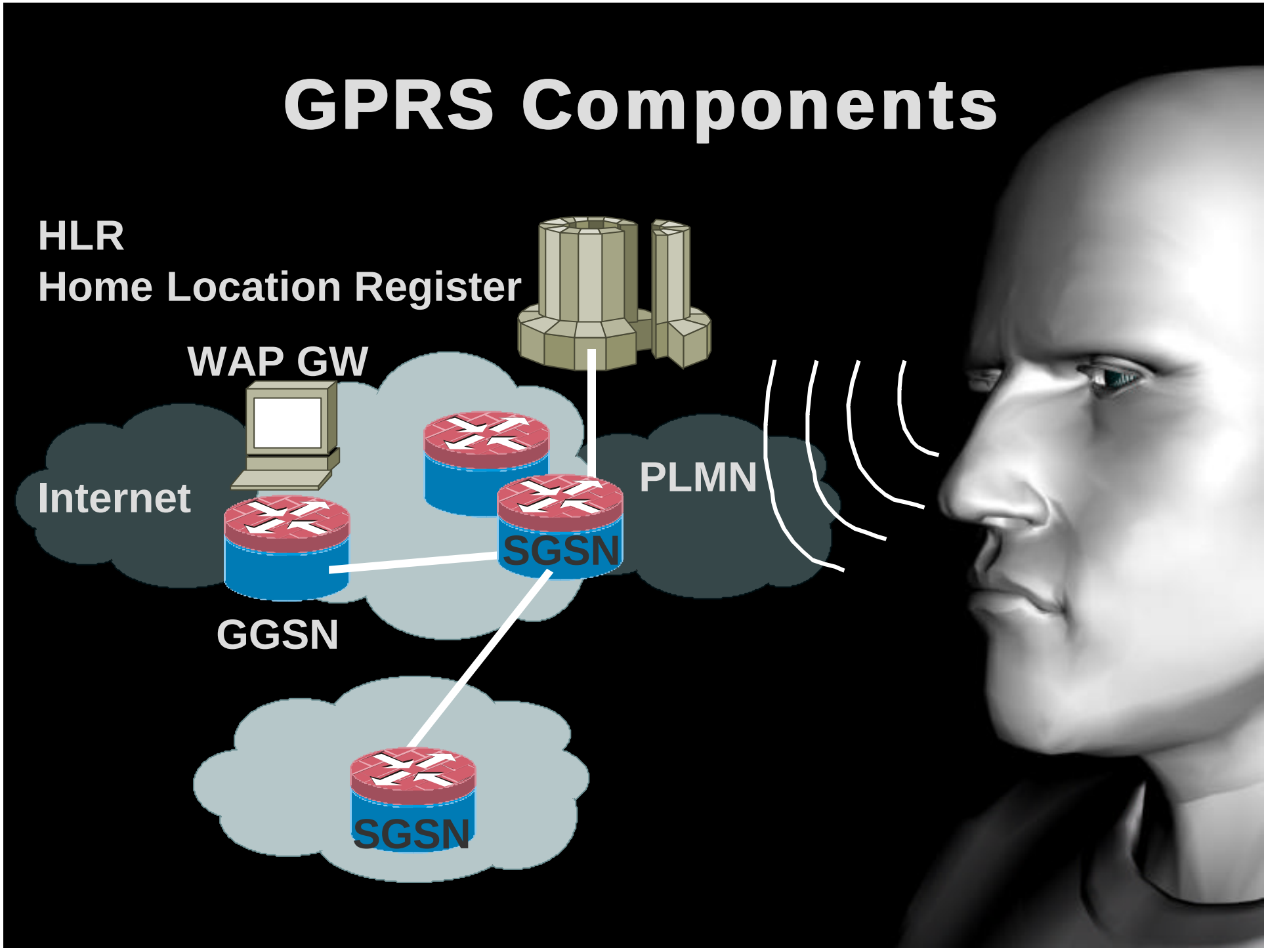
GSM Basics

- ***G***lobal ***S***ystem for ***M***obile communications –
  Mobile Phone Network
- Authentication based on key material on SIM card and on the network
- GSM core network relies on caller identification via MSISDN and key material for authentication and billing
- Caller spoofing generally considered hard to do

GSM Basics – GPRS

- General Packet Radio Service
- Packet oriented data transfer for mobile devices
- Backbone build using TCP/IP
- Authentication via GSM and optional PPP (CHAP,PAP)
- Network access, routes and IP addressing depending on Access Point Name (APN)

GPRS Components



GPRS Attack Points

- The GGSN is just another TCP/IP device facing the Internet
  - @stake: Nokia GPRS 1 IPSO DoS with TCP Option 0xFF
- APN guessing (WarAPNing?)
  - APNs often selected by company name for mobile VPNs
  - APN filtering in HLR possible but rarely implemented
  - APNs are not considered a secret ☺

Backbone hacks: GTP

- GPRS Tunneling Protocol (ETSI TS 129 060)
- Transports user protocols in the GPRS backbone
- Controls inter-network roaming
- Control channel and user data channel
- xGSN has to support all old protocol versions from 0 on

What‘s so cool?

- Fully UDP based protocol
- No authentication
- Full control over all GPRS nodes
- Creation, update or deletion of user contexts and tunnels
- Rerouting, redirection and relocation control
- Context billing configuration
- „Invitation“ of mobile users to APNs
- Forced roaming of users

Anonymous HTTP

- Wireless Application Protocol uses
  - Wireless Transport Protocol (WTP)
  - Wireless Session Protocol (WSP)
- WSP uses connections over UDP
  - Not easily spoofable due to 32bit session IDs
- WSP supports connectionless transactions
  - No acknowledge messages
  - Single UDP packet transfer
  - Full HTTP Request capabilities

Anonymous HTTP [2]

1. GPRS WAP APN or open WAP Gateway
2. Send HTTP request to WAP Gateway using someone else‘s IP address
3. Destination UDP port 9200
4. Enjoy

Note: Victim IP addresses can be collected by running a WAP site using HTTP header field „X-Forwarded-For“.

Siemens S55 – Bluetooth

- Pairing to death
  - Every connection creates a dialog
  - Connection structures not cleared
  - Bluetooth connection exhaustion
  - All you need is l2ping
- The Big Inbox
  - S55 accepts any file sent to it
  - Sending 2100 files is possible
  - Deleting 2100 files is not

Siemens S55 – Java

- NULL Pointer Exceptions
  - Generally crash JVM (white screen of death)
- JAD file parsing
  - Missing „MIDlet-Name“ and „MIDlet-1“ tags crash JVM
  - Overlong „MIDlet-Name“ leads to buffer overflow in filesystem

Siemens S55 – Spy.jar

- Outgoing WAP connections
  - Java Applications can connect to WAP sites without user permission
  - Com.siemens.mp.gsm. PhoneBook.getMDN() allows access to missed call list
  - System.getProperty(„IMEI“) returns the International Mobile Equipment Identity

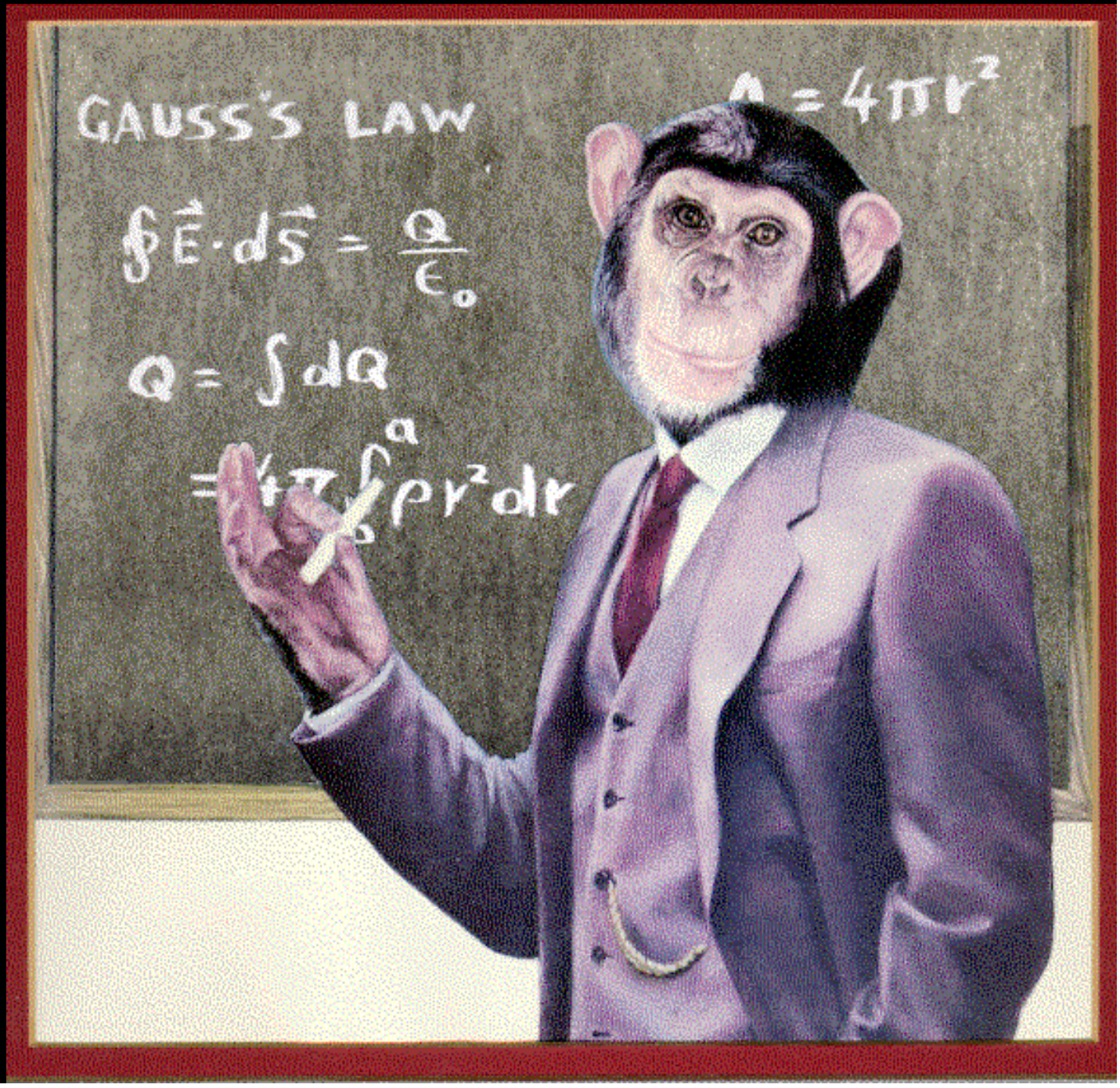
Siemens S55 – Time.jar

- Sending SMS or placing calls via Java applications requires user permission
- Permission is obtained via dialog
- Filling the screen obscures the dialog
- User answers a different question
- Outgoing call triggered but terminates Java application

Siemens S55 GIFt

- S55 supports GIF for MMS
- GIF file format has a Virtual Screen section
- Changing the virtual screen offset for a picture or one frame in an animated GIF crashes the device
- Placing such a pic as background renders the device unusable

And
Now
For
Something
Completely
Different



A small bug …

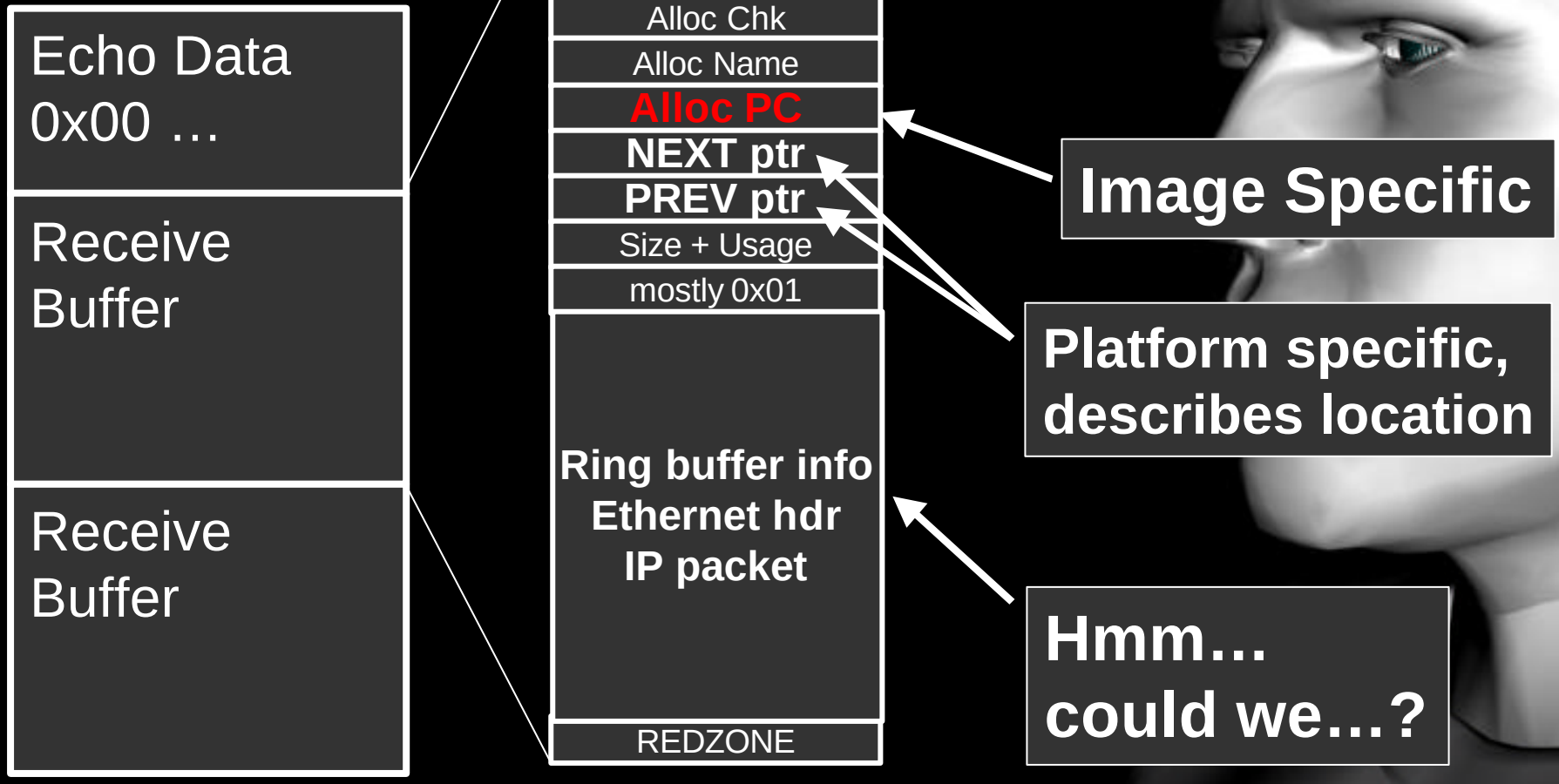
- Cisco IOS 11.x and below
- UDP Echo service memory leak
  - Device sends as much data back to the sender as the UDP length field said it got
  - Leaks IO memory blocks
  - IO memory contains actual packet data – and not just ours
  - We are talking about 19kbytes here
- Comparable bug surfaced in IOS 12.x Cisco Express Forwarding (CEF) code

IOS Fingerprinting

- Leaked IO memory contains memory block headers
  - Block headers contain address of who allocated the block
  - Address of allocating function changes per image
  - Address range changes per platform
- Result:
  Reliable remote IOS fingerprint

IOS Fingerprinting [2]

In detail:



Remote IOS Sniffing

- Leaked IO memory contains packets in the receive buffers (RX ring ds elements)
- Phenoelit IOSniff
  - Repeated memory leak retrieval
  - Memory block identification
  - Packet offset identification
  - Packet decoding
  - Caching and duplicate prevention

IOS HTTP bug

- Almost all embedded HTTP implementations are vulnerable – Cisco is no exception
- Integer or counting related issue
- IOS 11.x – 12.2.x
- Requires sending of a 2GB sized URL to the device
- Stack based buffer overflow

Exploitation – issues in the past

- For heap overflows, we need several image and configuration depend addresses
  - PREV pointer in the memory block
  - Size value in IO memory exploitation
  - Stack location
  - Own code location
- Requirements made reliable remote exploitation hard / impossible

What we got now

- UDP Echo memory leak
  - Attacker provided binary data (the delivered Echo content)
  - Live IOS memory addresses (leaked IO memory block headers)
  - Ability to fill multiple memory areas with our binary data (Ring buffer)
- HTTP Overflow
  - Direct frame pointer and return address overwrite

What we can do now

- Send full binary shell code
- Calculate the address of the code using IO memory block header information
- Select the shell code that is most likely not modified
- Directly redirect execution in the provided shell code
- Own the box

Combining

1. Send the maximum URL length allowed by IOS
2. Send 2GB of additional URL elements in correct sized chunks
3. Perform UDP memory leak several times with shell code in the request packet
4. Make intelligent decision on which address to use
5. Complete overflow and gain control

Again, in color

HTTP Connect + legal size URL

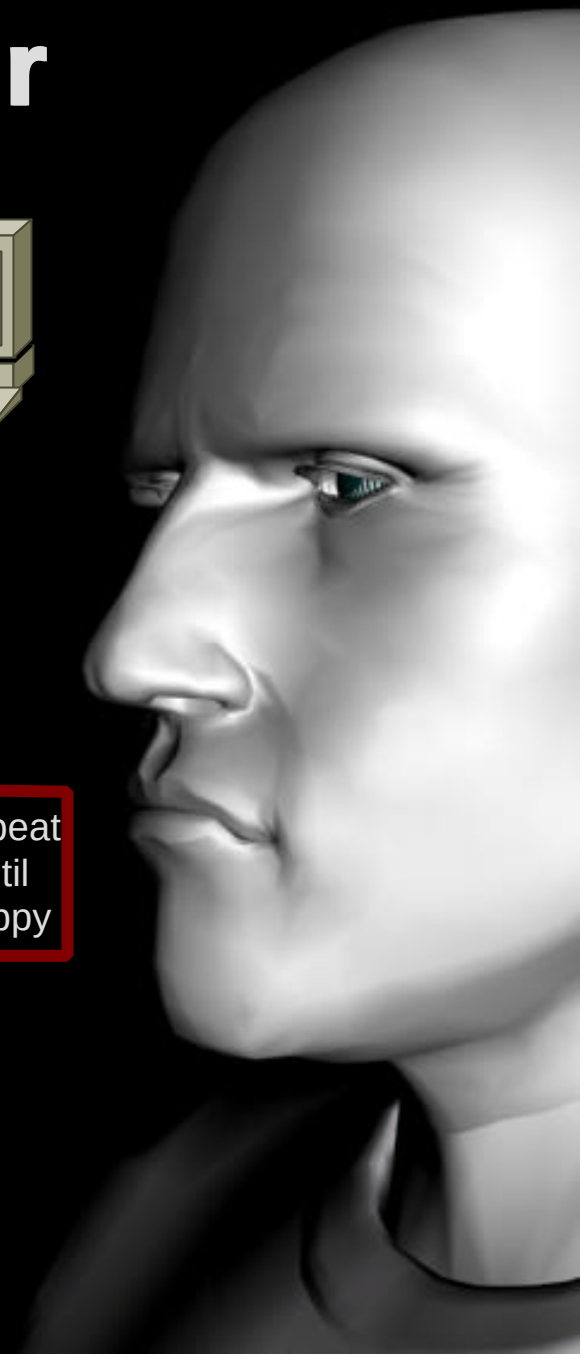
2GB of /AAAAAA/AAA..../

Shell code to UDP Echo

Repeat until happy

Leaked memory back

Complete HTTP overflow



Binary via HTTP

- Cisco‘s HTTP doen‘t like all characters
  - Slash , 0x0a, 0x0d and 0x00 are obviously bad for HTTP
  - Some others are bad as well
- HTTP encoding (%XY) supported
- Decoding seems to take place in the exact same buffer
- Return address HTTP encoded

Return address selection

- Several address selection strategies tested
  - Last address obtained (about 50% success)
  - Randomly selected address (about 50%-60% success)
  - Highest memory location (about 0%-10% success)
  - Lowest memory location (about 90% success)
  - Most frequently seen address (about 30%-40% success)

Cisco shell code in the past

- Complete configuration replacement in NVRAM and requires reboot
- Required knowledge
  - Attacked interface
  - IP address
  - Basic routing information
- Looses information of original configuration
  - Passwords and keys
  - Other routing information
  - Access lists
  - Logging information

Researching binary IOS

- Cisco supports serial gdb
- ROM Monitor (rommon) allows limited debugging
  - Breakpoints
  - Watchpoints
  - Disassembly
- Code identification simple
  - Related debug strings can be found in the code
  - Data and text segment are intermixed with each other
  - Strings stored before the related function

Next generation code

- Runtime IOS patching
- Patched (disabled) elements:
  - IOS text segment checksum function
  - Authentication requirement for incoming VTY connections
  - Verification return code from “enable mode” function
  - In the future: ACLs or BGP neighbor check?
- Keep IOS running … but how?

Clean return

- Overflow destroys significant amounts of stack due to HTTP encoding
  - 24 bytes encoded: **%fe%fe%ba%be%f0%0d%ca%fe**
  - 8 bytes decoded
- Motorola call structure uses frame pointer in A6 and saved stack pointer on stack
- Moving the stack pointer before the saved SP of any function restores SP and A6
- Search stack “upward” for return address of desired function

```
SP = <current> - 4
unlk a6
rts
```

Clean return code

IOS 11.3(11b) HTTP overflow find-return code

```
   move.l    a7,a2
findret:
   addq.l    #0x01,a7
   cmp.l     #0x0219fcc0,(a7)
   bne       findret
   move.l    a7,(a2)
   sub.l     #0x00000004,(a2)
   move.l    (a2),a6
   clr.l     d0
   movem.l   -4(a6),a2
   unlk      a6
   rts
```

Runtime IOS patching

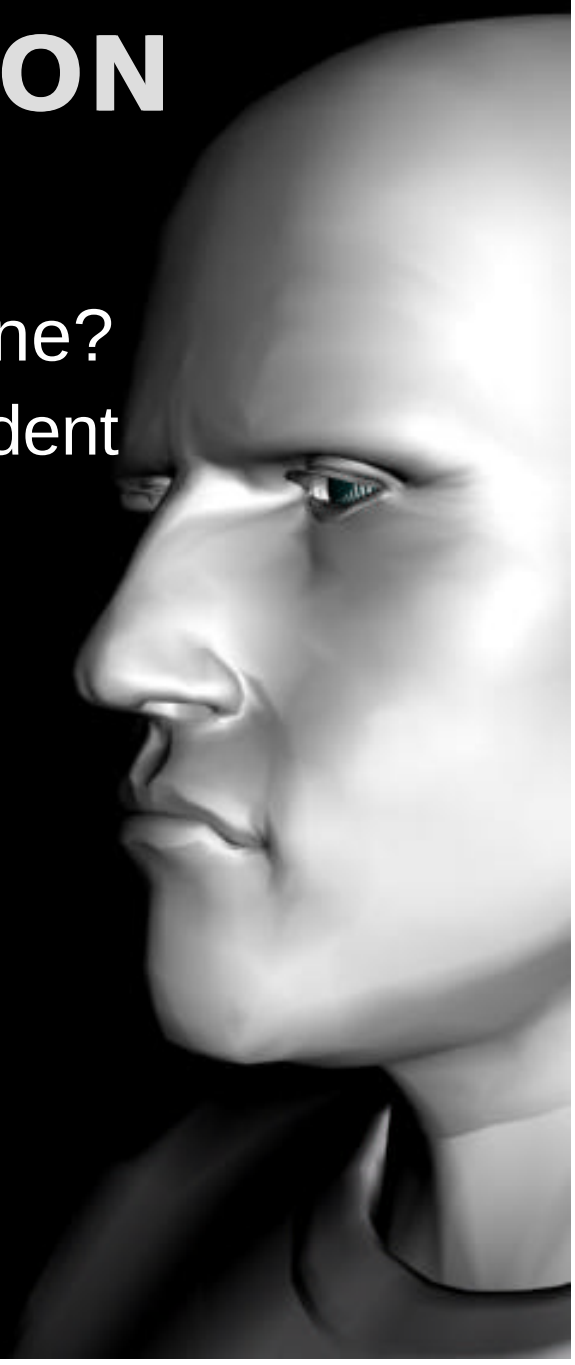
- Advantages
  - Router stays online
  - Configuration preserved
  - Backdoor in IOS runtime code
- Disadvantages
  - Depending on image
  - Large target list required (code addresses per image)
  - Annoying “checksum error” message on console ☺

CISCO CASUM EST

- Reliable remote IOS exploitation
- Address calculation and shell code placement via UDP Echo info leak
- Address selection using second smallest address
  - first used for HTTP transfer itself
- Runtime IOS patch disables VTY and enable mode password verification

Tribute to DEFCON

- Image independent shell code anyone?
  - Modifying IOS code is image dependent
  - Modifying IOS configuration is not
- Runtime config modification code preserves original config and changes only a few „elements“.
- Shell code needs
  - Strstr()
  - Memcpy()
  - Checksum()
- Well, we can do that ☺



Config modification code

- Find beginning of configuration in NVRAM
- Find occurrences of
  “\n password “
  “\nenable “
- Replace occurrences with your “data”
- Hereby replace authentication information for
  - Console passwords
  - VTY line passwords
  - Enable passwords
  - Enable secrets
- Recalculate checksum
- Reboot

Config modification code

- Advantages
  - Image independent
  - Configuration preserved
  - More choices of what to do
- Disadvantages
  - Depending on platform
  - Router has to reboot once

So what?

- Didn’t we mention, that …
  - … you shouldn’t run unneeded services
  - … you should protect your infrastructure
  - … you should not copy data into buffers that are not large enough to hold it
- IOS moves “forward”
- Legal interception is build into IOS

  “My other computer is your legal interception system.”
  – Hey Jaya, it’s no longer a joke.
- If your infrastructure is owned, you can’t defend your systems.
- Other people exploit IOS as well – only we do it in public.

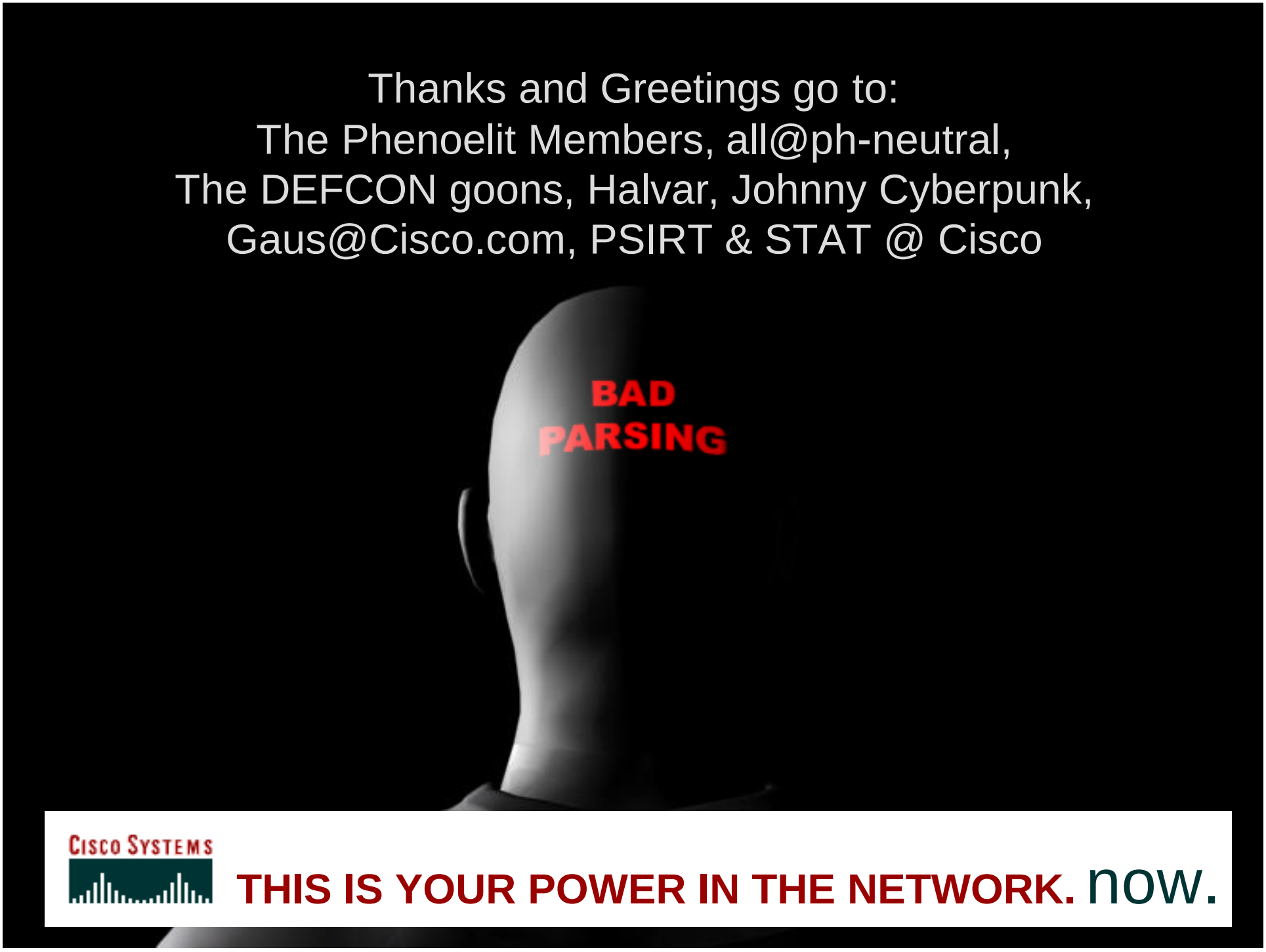
Defense – Mobile Phones

- Turn off unneeded interfaces on your phone
  - Bluetooth
  - IRDA
- Do not run Java code you don’t trust
- When receiving files, delete them directly instead of opening them (remember your Outlook)
- Keep your Phone firmware up to date
- Don’t use GPRS based VPN solutions

Defense - Cisco

- Do not trust devices, just because they are in a black box
- Keep your IOS up to date
- If possible, block direct communication to your infrastructure devices
- Do not run unneeded services on routers and other devices
- Prefer out-of-band management
- Include your router in your IDS watch list

Thanks and Greetings go to:
The Phenoelit Members, all@ph-neutral,
The DEFCON goons, Halvar, Johnny Cyberpunk,
Gaus@Cisco.com, PSIRT & STAT @ Cisco



Cisco Systems
THIS IS YOUR POWER IN THE NETWORK. now.