Columbia University

in the City of New York

Killing the Myth of Cisco IOS Diversity

Towards Large-Scale Exploitation of Cisco IOS

Ang Cui
ang@cs.columbia.edu
Columbia University Intrusion Detection Systems Lab

Prof. Salvatore J. Stolfo | sal@cs.columbia.edu
Jatin Kataria | jk3319@columbia.edu

Columbia University
in the City of New York

BlackHat Briefings USA 8.3.2011

Prior Work

FX, 2003
Lynn, 2005
Uppal, 2007
Davis, 2007
Muniz, 2008
FX, 2009
Muniz and Ortega, 2011

Not comprehensive, but is a good start

Motivation

Motivation

Cisco IOS is a high value target

Motivation

Cisco IOS is a high value target

Cisco IOS is "undefended"

Motivation

Cisco IOS is a high value target

Cisco IOS is "undefended"

Cisco IOS is "unmonitored"

Motivation

Cisco IOS is a high value target

Cisco IOS is "undefended"

Cisco IOS is "unmonitored"

Cisco IOS can be **exploited**, just like everything else

Motivation

But there the problem of **software diversity**

Motivation

But there the problem of **software diversity**

Approximately 300,000 unique IOS images
No reliable binary invariant

Motivation

But there the problem of **software diversity**

Approximately 300,000 unique IOS images
No reliable binary invariant

The (last) major obstacle in large-scale IOS exploitation

Reliable Shellcode

- IOS Diversity means **Binary** Diversity

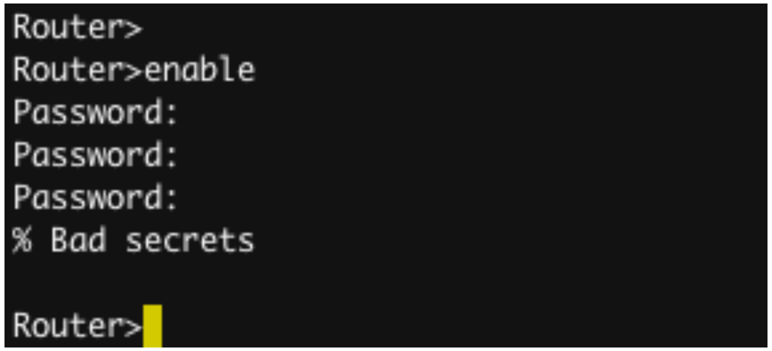
Reliable Shellcode

- IOS Diversity means Binary Diversity, not **functional** diversity

Killing the Myth of Cisco IOS Diversity

Reliable Shellcode

- IOS Diversity means Binary Diversity, not functional diversity
- In fact, IOS is rich in **functional invariants**
  - For example:



```
Router>
Router>enable
Password:
Password:
Password:
% Bad secrets

Router>
```

Functional monoculture in every box!

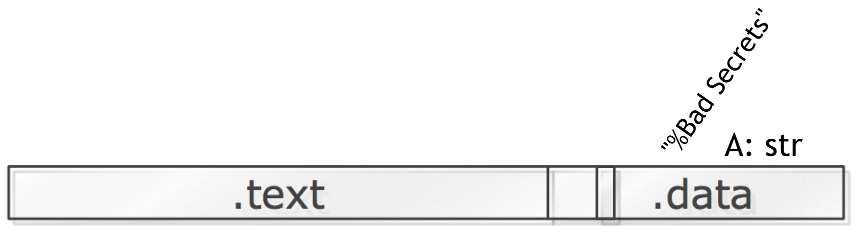
RELIABLE SHELLCODE

- GENERAL STRATEGY TO OVERCOME IOS DIVERSITY
  - USE FUNCTIONAL INVARIANTS TO RESOLVE BINARY TARGETS
  - FOR EXAMPLE: (SEE FX, 2009)

Killing the Myth of Cisco IOS Diversity

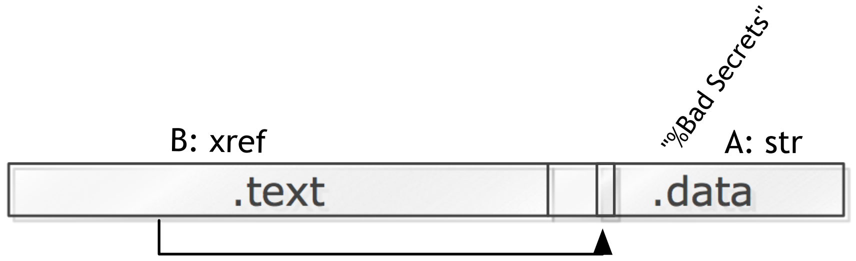
Reliable Shellcode

- General strategy to overcome IOS diversity
  - Use functional invariants to resolve binary targets
  - For example: (see FX, 2009)



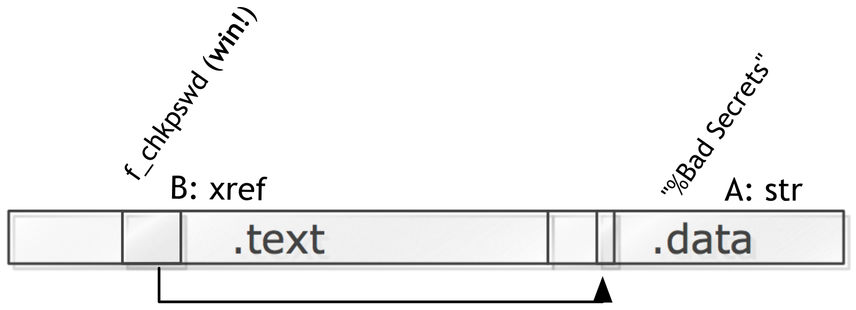
Reliable Shellcode

- General strategy to overcome IOS diversity
    - Use functional invariants to resolve binary targets
    - For example: (see FX, 2009)



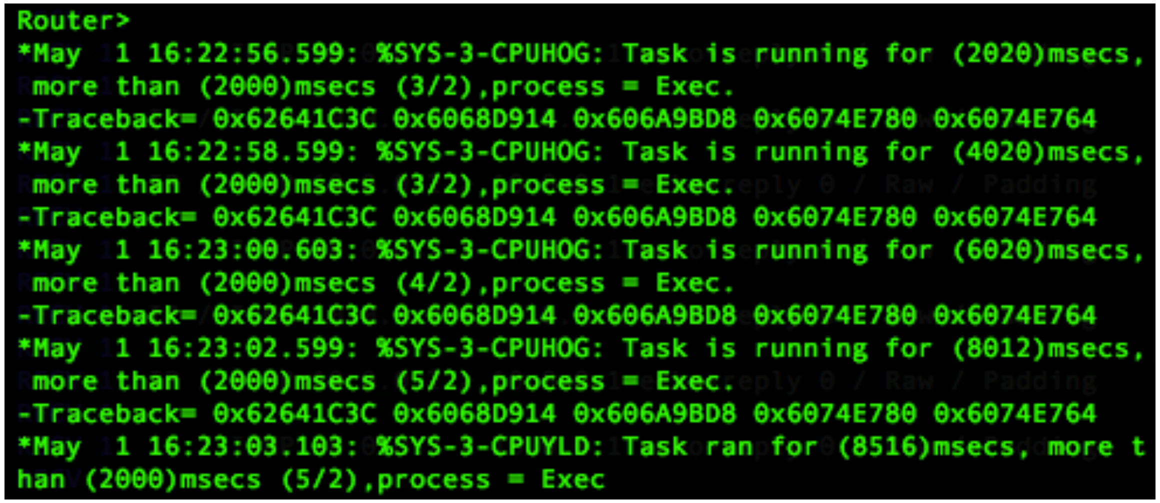
RELIABLE SHELLCODE

- GENERAL STRATEGY TO OVERCOME IOS DIVERSITY
  - USE FUNCTIONAL INVARIANTS TO RESOLVE BINARY TARGETS
  - FOR EXAMPLE: (SEE FX, 2009)



DISASSEMBLING SHELLCODE #1

- THERE IS A CATCH (CALLED THE **WATCHDOG TIMER**)



```
Router>
*May  1 16:22:56.599: %SYS-3-CPUHOG: Task is running for (2020)msecs,
 more than (2000)msecs (3/2),process = Exec.
-Traceback= 0x62641C3C 0x6068D914 0x606A9BD8 0x6074E780 0x6074E764
*May  1 16:22:58.599: %SYS-3-CPUHOG: Task is running for (4020)msecs,
 more than (2000)msecs (3/2),process = Exec.
-Traceback= 0x62641C3C 0x6068D914 0x606A9BD8 0x6074E780 0x6074E764
*May  1 16:23:00.603: %SYS-3-CPUHOG: Task is running for (6020)msecs,
 more than (2000)msecs (4/2),process = Exec.
-Traceback= 0x62641C3C 0x6068D914 0x606A9BD8 0x6074E780 0x6074E764
*May  1 16:23:02.599: %SYS-3-CPUHOG: Task is running for (8012)msecs,
 more than (2000)msecs (5/2),process = Exec.
-Traceback= 0x62641C3C 0x6068D914 0x606A9BD8 0x6074E780 0x6074E764
*May  1 16:23:03.103: %SYS-3-CPUYLD: Task ran for (8516)msecs, more t
han (2000)msecs (5/2),process = Exec
```

COMPUTE TOO LONG, AND YOU WILL GET CAUGHT!

SHELLCODE IS HEAVILY **RESOURCE** CONSTRAINED,.

MUST RESOLVE BINARY TARGET USING FAST, (SUB)LINEAR ALGORITHMS.

Interrupt-Hijack Shellcode

- Let's kill 3 birds with one stone

Interrupt-Hijack Shellcode

- Let's kill 3 birds with one stone
    - Faster
        - enable-bypass shellcode: 2N algorithm
        - Interrupt-hijack shellcode: twice as fast

Interrupt-Hijack Shellcode

- Let' s kill 3 birds with one stone
    - Faster
    - Stealthier
        - Enable-bypass, vty rebind, etc requires persistent TCP connection
        - Interrupt-Hijack uses the payload of process-switched packets as a covert command and control channel
        - C&C is bidirectional thanks to IOMEM scrubber

Interrupt-Hijack Shellcode

- Let's kill 3 birds with one stone
  - Faster
  - Stealthier
  - More Control
    - No need to be constrained by IOS shell
    - Rootkit runs @ supervisor mode. We can even write to EEPROM (See last slide)

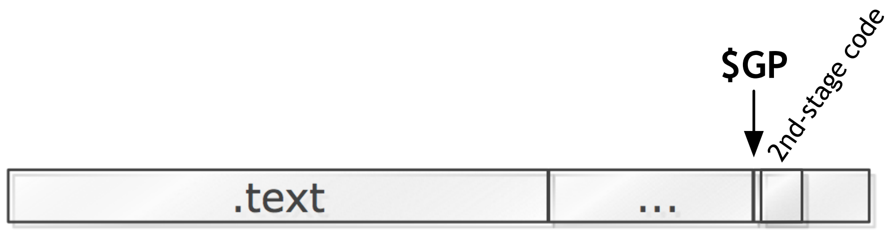
Killing the Myth of Cisco IOS Diversity

Interrupt-Hijack Shellcode

- 1ST STAGE:

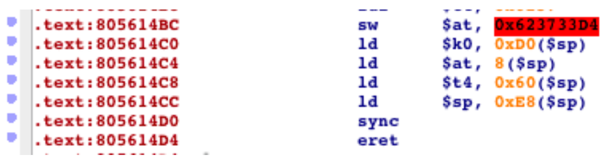
INTERRUPT-HIJACK SHELLCODE

- 1ST STAGE: UNPACK 2ND STAGE

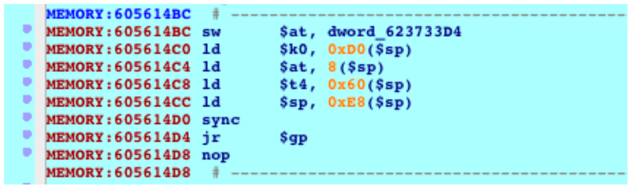


INTERRUPT-HIJACK SHELLCODE

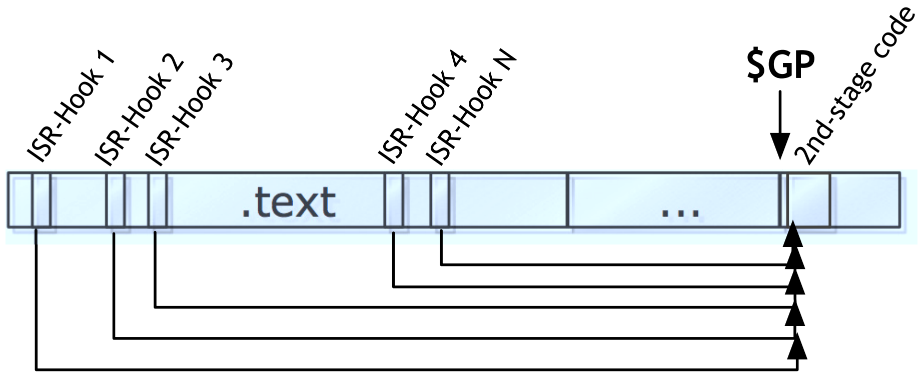
- 1ST STAGE: UNPACK 2ND STAGE, HIJACK ALL INT-HANDLERS



```
.text:805614BC                 sw      $at, 0x623733D4
.text:805614C0                 ld      $k0, 0xD0($sp)
.text:805614C4                 ld      $at, 8($sp)
.text:805614C8                 ld      $t4, 0x60($sp)
.text:805614CC                 ld      $sp, 0xE8($sp)
.text:805614D0                 sync
.text:805614D4                 eret
```



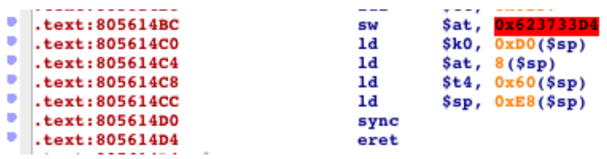
```
MEMORY:605614BC  # -------------------------------------------
MEMORY:605614BC  sw      $at, dword_623733D4
MEMORY:605614C0  ld      $k0, 0xD0($sp)
MEMORY:605614C4  ld      $at, 8($sp)
MEMORY:605614C8  ld      $t4, 0x60($sp)
MEMORY:605614CC  ld      $sp, 0xE8($sp)
MEMORY:605614D0  sync
MEMORY:605614D4  jr      $gp
MEMORY:605614D8  nop
MEMORY:605614D8  # -------------------------------------------
```



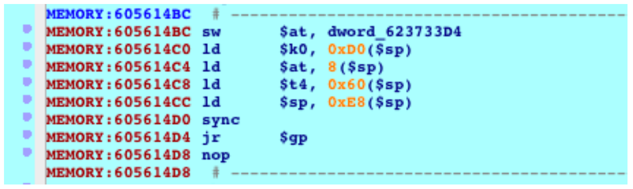
Killing the Myth of Cisco IOS Diversity

Interrupt-Hijack Shellcode

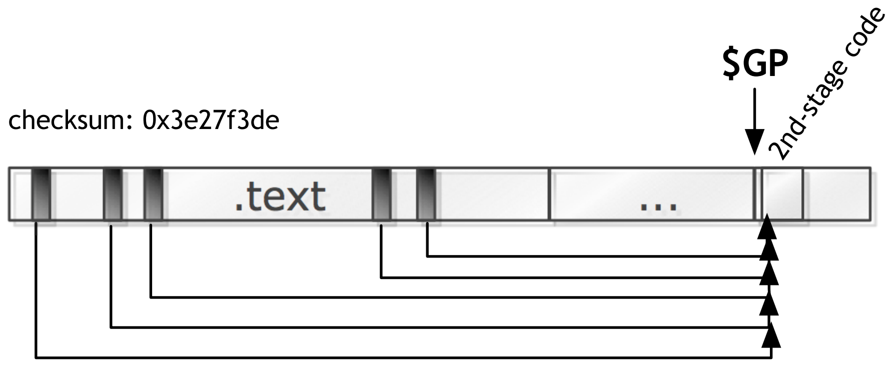
- 1ST stage: Unpack 2ND stage, hijack all int-handlers, compute **hash** on addresses of "ERET" instructions (**why?**)



```
.text:805614BC    sw      $at, 0x623733D4
.text:805614C0    ld      $k0, 0xD0($sp)
.text:805614C4    ld      $at, 8($sp)
.text:805614C8    ld      $t4, 0x60($sp)
.text:805614CC    ld      $sp, 0xE8($sp)
.text:805614D0    sync
.text:805614D4    eret
```

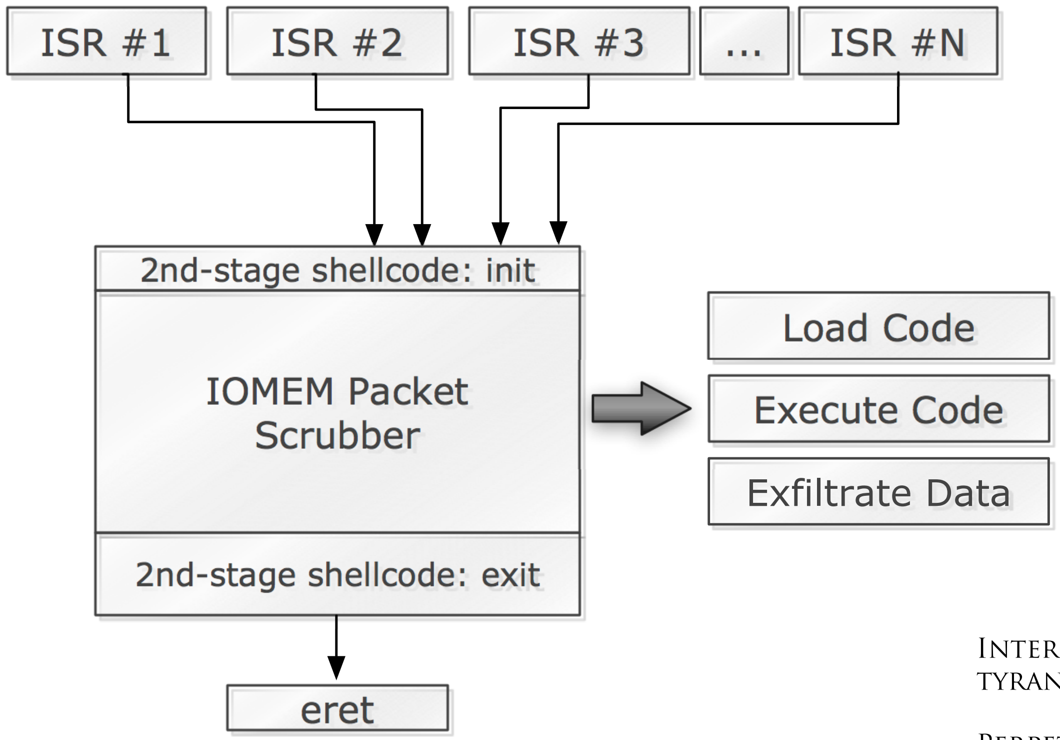


```
MEMORY:605614BC    # ----------------------------------------
MEMORY:605614BC sw      $at, dword_623733D4
MEMORY:605614C0 ld      $k0, 0xD0($sp)
MEMORY:605614C4 ld      $at, 8($sp)
MEMORY:605614C8 ld      $t4, 0x60($sp)
MEMORY:605614CC ld      $sp, 0xE8($sp)
MEMORY:605614D0 sync
MEMORY:605614D4 jr      $gp
MEMORY:605614D8 nop
MEMORY:605614D8    # ----------------------------------------
```



INTERRUPT-HIJACK SHELLCODE

- 2ND-STAGE: EXCEPTION HIJACK AND IOMEM SNOOPING

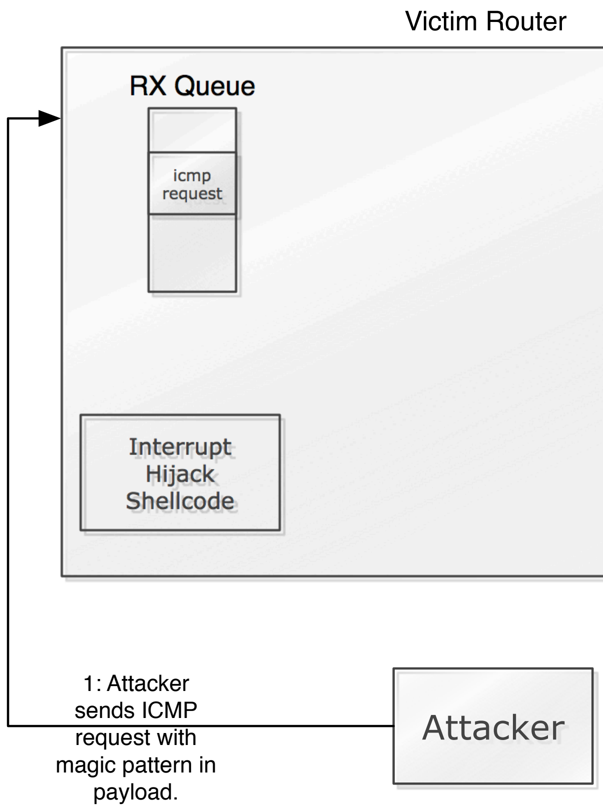


- THE (MIPS) ERET, OR EXCEPTION-RETURN IS AN ARCHITECTURE INVARIANT
- ISR **ENTRY** POINT IS A **BINARY** INVARIANT, TYPICALLY FOUND AT 0X600080180, ETC
- CAN JUST HIJACK ENTRY POINT, BUT THERE IS AN ULTERIOR MOTIVE
- USE ERET LOCATIONS IN THE IMAGE TO **FINGERPRINT** IOS VERSION

INTERRUPT-HIJACK SHELLCODE FREES US FROM THE TYRANNIES OF THE WATCHDOG TIMER.

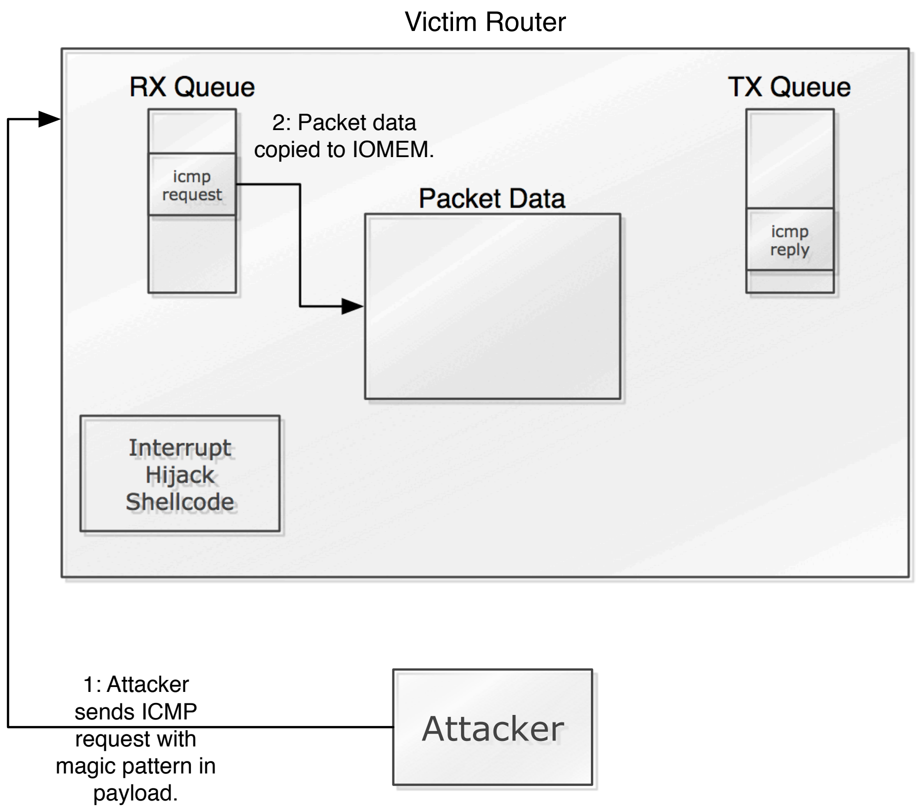
PERPETUAL, STEALTHY EXECUTION!

INT-HIJACK SHELLCODE: FINGERPRINT EXFILTRATION


Victim Router
RX Queue
icmp request
TX Queue
icmp reply
Interrupt Hijack Shellcode
Attacker
1: Attacker sends ICMP request with magic pattern in payload.

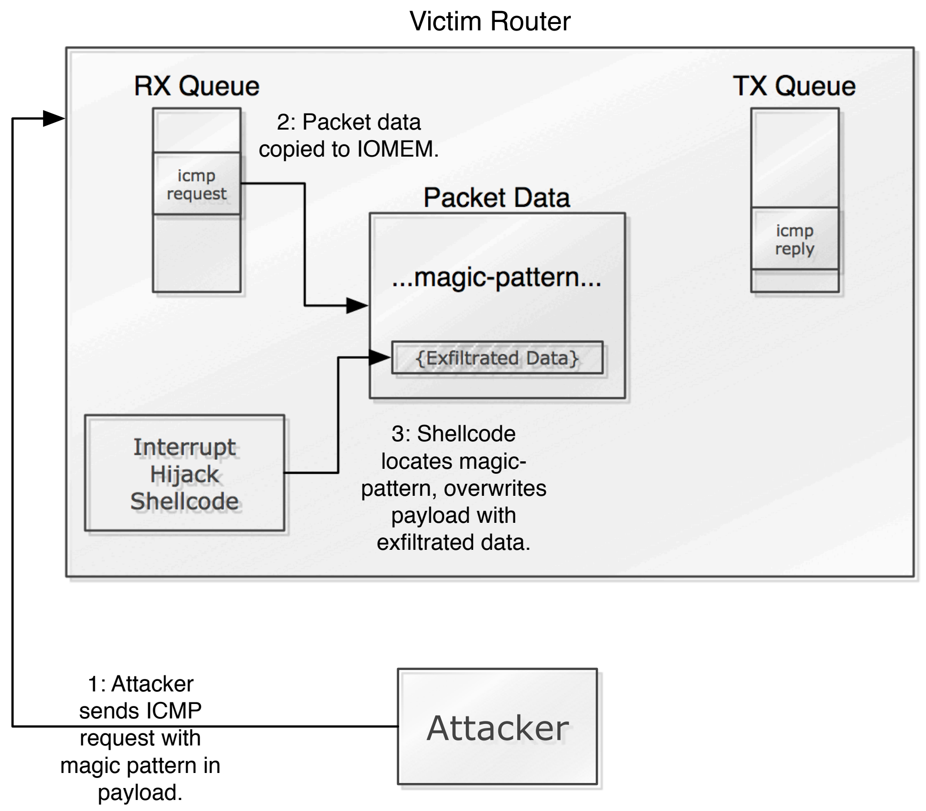
- ICMP IS CONVENIENT, BUT ANY "PROCESS-SWITCHED" PACKET WILL SUFFICE
- C&C INSIDE PAYLOAD OF "NORMAL" TRAFFIC
- COMPLEX THIRD-STAGE PAYLOADS CAN BE ASSEMBLED IN A "PROTOCOL-SPREAD-SPECTRUM" MANNER
- PING, DNS, PDUS, TCP, ALL THE SAME AS LONG AS IT IS PROCESS-SWITCHED

INT-HIJACK SHELLCODE: FINGERPRINT EXFILTRATION


Victim Router
RX Queue
icmp request
2: Packet data copied to IOMEM.
Packet Data
TX Queue
icmp reply
Interrupt Hijack Shellcode
1: Attacker sends ICMP request with magic pattern in payload.
Attacker

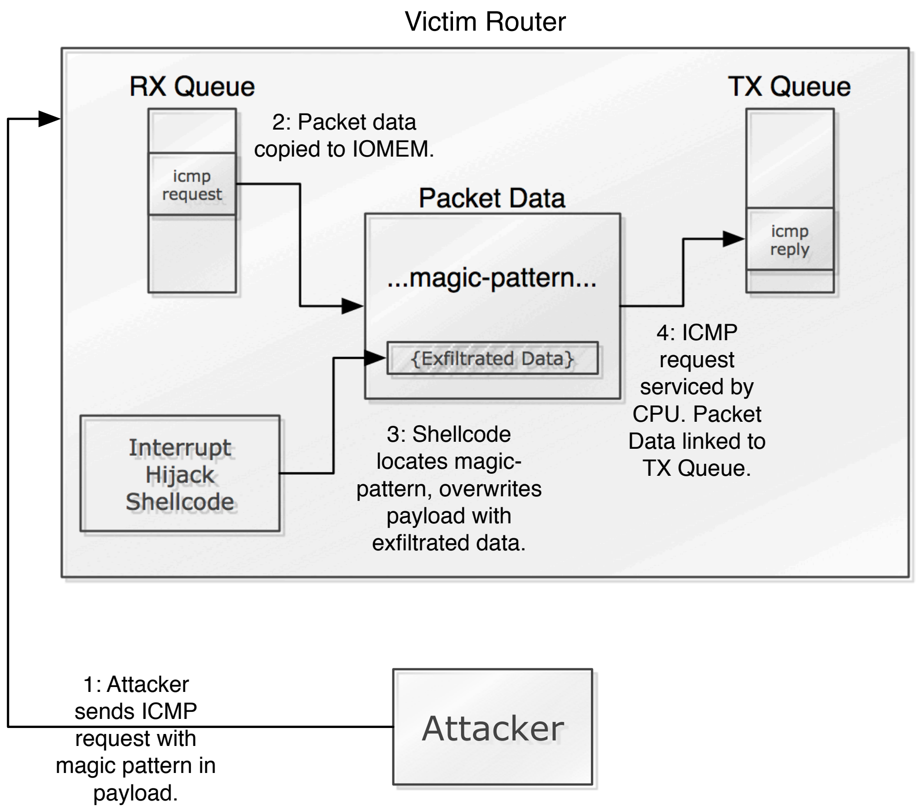
- ICMP IS CONVENIENT, BUT ANY "PROCESS-SWITCHED" PACKET WILL SUFFICE
- C&C INSIDE PAYLOAD OF "NORMAL" TRAFFIC
- COMPLEX THIRD-STAGE PAYLOADS CAN BE ASSEMBLED IN A "PROTOCOL-SPREAD-SPECTRUM" MANNER
- PING, DNS, PDUs, TCP, ALL THE SAME AS LONG AS IT IS PRCOESS-SWITCHED

INT-HIJACK SHELLCODE: FINGERPRINT EXFILTRATION


Victim Router
RX Queue
icmp request
2: Packet data copied to IOMEM.
Packet Data
...magic-pattern...
{Exfiltrated Data}
TX Queue
icmp reply
Interrupt Hijack Shellcode
3: Shellcode locates magic-pattern, overwrites payload with exfiltrated data.
1: Attacker sends ICMP request with magic pattern in payload.
Attacker

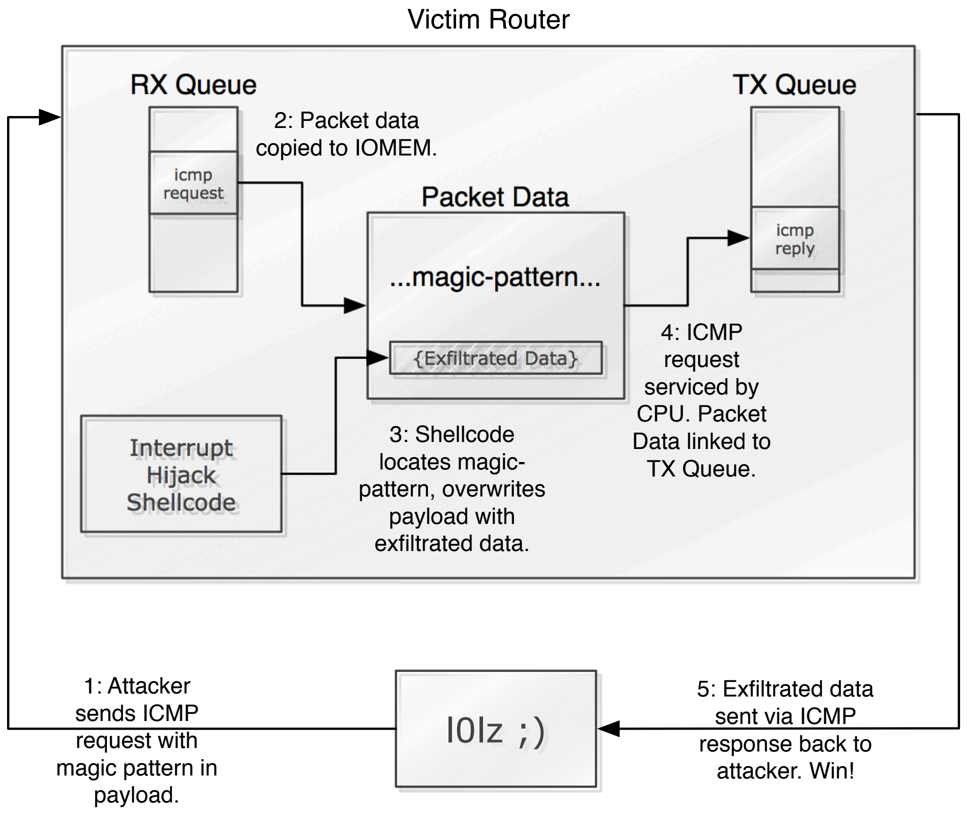
- ICMP IS CONVENIENT, BUT ANY "PROCESS-SWITCHED" PACKET WILL SUFFICE
- C&C INSIDE PAYLOAD OF "NORMAL" TRAFFIC
- COMPLEX THIRD-STAGE PAYLOADS CAN BE ASSEMBLED IN A "PROTOCOL-SPREAD-SPECTRUM" MANNER
- PING, DNS, PDUS, TCP, ALL THE SAME AS LONG AS IT IS PRCOESS-SWITCHED

INT-HIJACK SHELLCODE: FINGERPRINT EXFILTRATION


Victim Router
RX Queue
icmp request
2: Packet data copied to IOMEM.
Packet Data
...magic-pattern...
{Exfiltrated Data}
TX Queue
icmp reply
4: ICMP request serviced by CPU. Packet Data linked to TX Queue.
Interrupt Hijack Shellcode
3: Shellcode locates magic-pattern, overwrites payload with exfiltrated data.
1: Attacker sends ICMP request with magic pattern in payload.
Attacker

- ICMP IS CONVENIENT, BUT ANY "PROCESS-SWITCHED" PACKET WILL SUFFICE
- C&C INSIDE PAYLOAD OF "NORMAL" TRAFFIC
- COMPLEX THIRD-STAGE PAYLOADS CAN BE ASSEMBLED IN A "PROTOCOL-SPREAD-SPECTRUM" MANNER
- PING, DNS, PDUs, TCP, ALL THE SAME AS LONG AS IT IS PRCOESS-SWITCHED

INT-HIJACK SHELLCODE: FINGERPRINT EXFILTRATION


Victim Router
RX Queue
icmp request
2: Packet data copied to IOMEM.
Packet Data
...magic-pattern...
{Exfiltrated Data}
TX Queue
icmp reply
4: ICMP request serviced by CPU. Packet Data linked to TX Queue.
Interrupt Hijack Shellcode
3: Shellcode locates magic-pattern, overwrites payload with exfiltrated data.
1: Attacker sends ICMP request with magic pattern in payload.
I0lz ;)
5: Exfiltrated data sent via ICMP response back to attacker. Win!

- ICMP IS CONVENIENT, BUT ANY "PROCESS-SWITCHED" PACKET WILL SUFFICE
- C&C INSIDE PAYLOAD OF "NORMAL" TRAFFIC
- COMPLEX THIRD-STAGE PAYLOADS CAN BE ASSEMBLED IN A "PROTOCOL-SPREAD-SPECTRUM" MANNER
- PING, DNS, PDUS, TCP, ALL THE SAME AS LONG AS IT IS PRCOESS-SWITCHED

RUNTIME FINGERPRINT GIVES US POSITIVE ID ON THE VICTIM ROUTER'S HARDWARE PLATFORM AND IOS VERSION!

Reliable Shellcode

- General strategy to overcome IOS diversity
    - Use functional invariants to resolve binary targets
    - IOS Diversity is (very) finite
        - How do you defeat address space randomization?

Reliable Shellcode

- General strategy to overcome IOS diversity
  - Use functional invariants to resolve binary targets
  - IOS Diversity is (very) finite
    - How do you defeat ASR if there are **ONLY** 300,000 possible permutations?

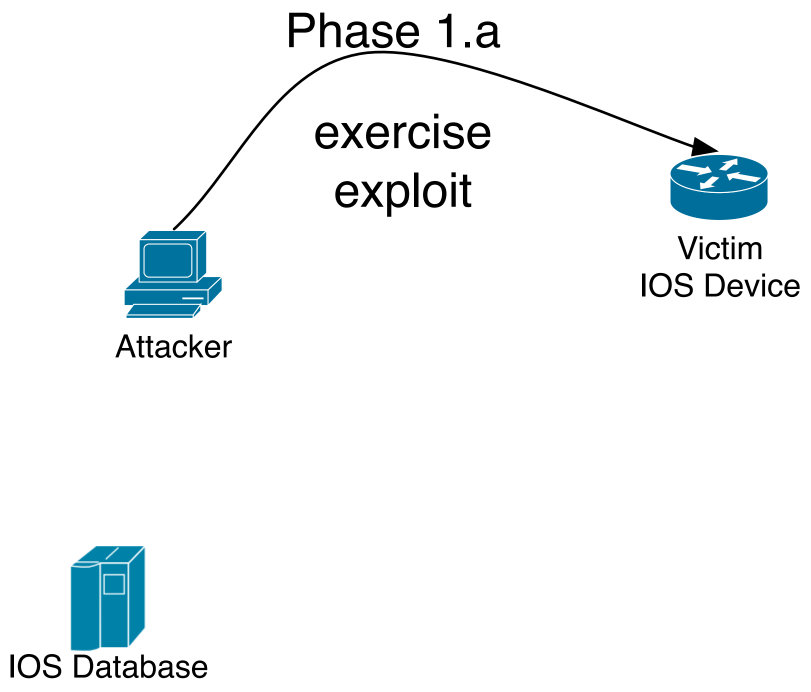
Killing the Myth of Cisco IOS Diversity

Reliable Shellcode

- General strategy to overcome IOS diversity
  - Use functional invariants to resolve binary targets
  - IOS Diversity is (very) finite
    - How do you defeat ASR if there are ONLY 300,000 possible permutations?
    - Build a lookup table!

Killing the Myth of Cisco IOS Diversity

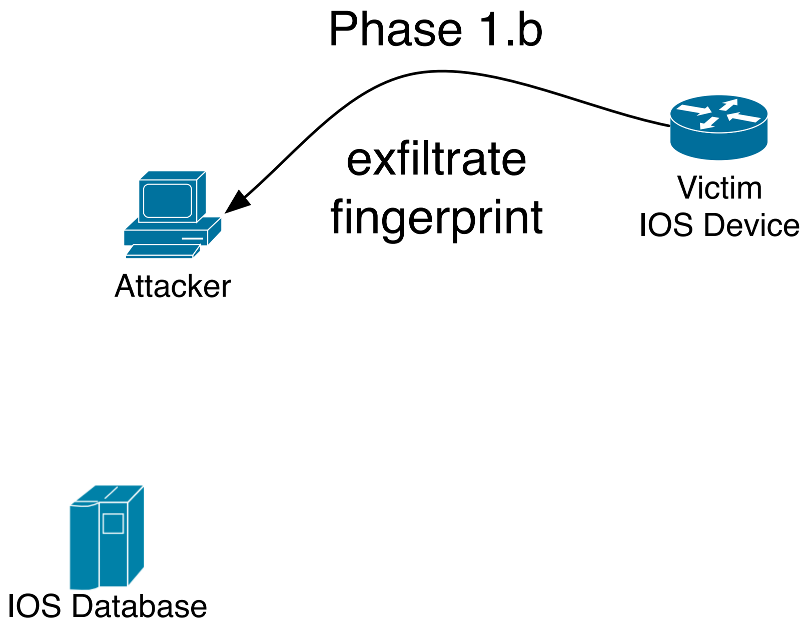
Generalized reliable exploitation of IOS (in 4 simple steps)



1.a: exploit vulnerability, load and run 1st stage eret-hijack rootkit (~400 bytes, PIC, will run anywhere)

Killing the Myth of Cisco IOS Diversity

Generalized reliable exploitation of IOS (in 4 simple steps)

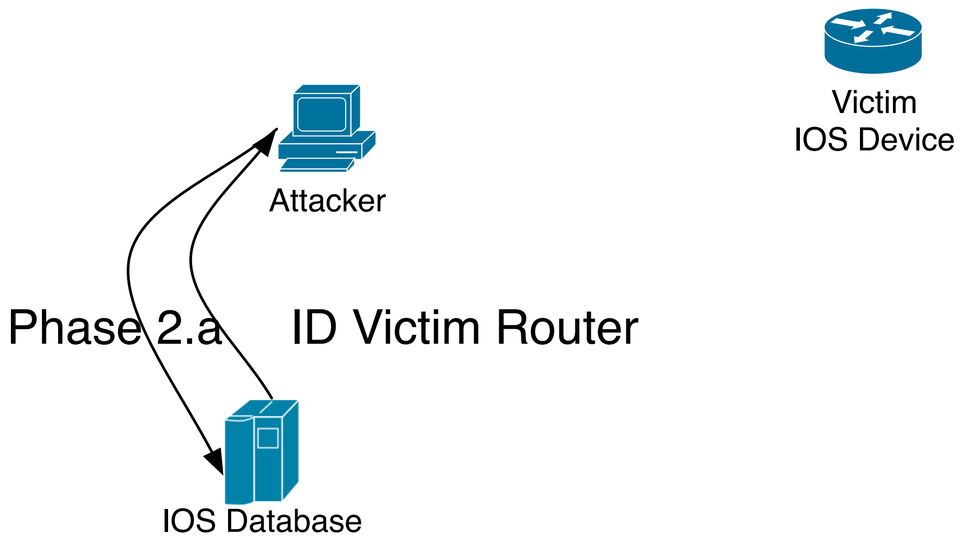


1.a: Exploit vulnerability, load and run 1st stage ERET-hijack rootkit (~400 bytes, PIC, will run anywhere)

1.b: 1st stage code locates/hijacks all ERET instructions, exfiltrate hash (**fingerprint**) of ERET-addrs back to attacker (via ICMP, etc)

Killing the Myth of Cisco IOS Diversity

Generalized reliable exploitation of IOS (in 4 simple steps)



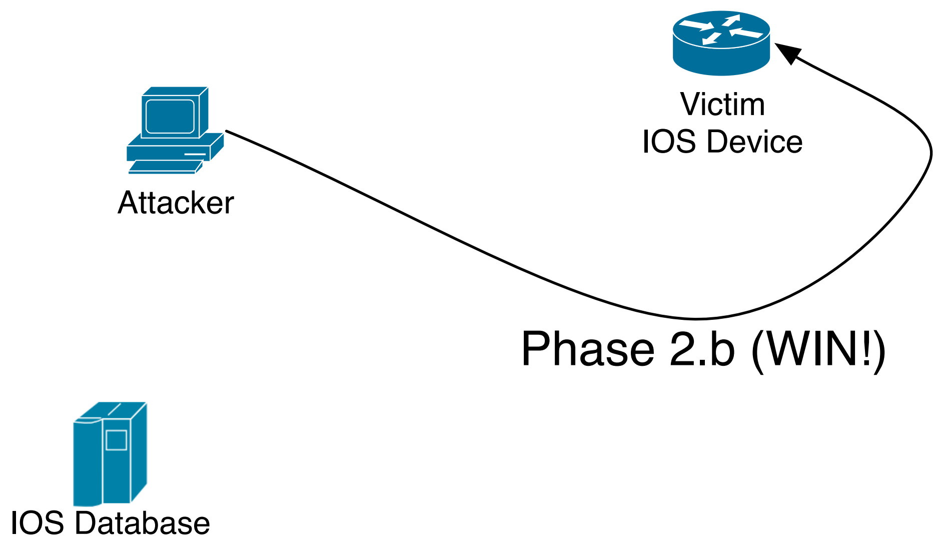
1.a: Exploit vulnerability, load and run 1st stage ERET-hijack rootkit (~400 bytes, PIC, will run anywhere)

1.b: 1st stage code locates/hijacks all ERET instructions, exfiltrate hash (**fingerprint**) of ERET-addrs back to attacker (via ICMP, etc)

2.a: Attacker consults offline IOS fingerprint database, makes positive ID (hardware platform, IOS version)

Killing the Myth of Cisco IOS Diversity

Generalized reliable exploitation of IOS (in 4 simple steps)



1.A: Exploit vulnerability, load and run 1st stage ERET-hijack rootkit (~400 bytes, PIC, will run anywhere)

1.B: 2st stage code locates/ hijacks all ERET instructions, exfiltrate hash (**fingerprint**) of ERET-addrs back to attacker (via ICMP, etc)

2.A: Attacker consults offline IOS fingerprint database, makes positive ID (hardware platform, IOS version)

2.B: Construct version dependent 3rd stage payload. Upload using 2nd stage C&C (again, using ICMP, etc)... **WIN**!

3rd Stage Payloads!

- More demos
- Third-stage payloads to:
    - Disable IOS integrity verification command "show sum"
    - Disable password authentication

    - Remote Bricking of router motherboard

Killing the Myth of Cisco IOS Diversity

Sacrifice to the Demo Gods

Remotely bricking router using 3rd-stage payload over ICMP!

What's Next (Offensive)?

- More comprehensive fingerprint database
    - ~3,000 images in the fingerprint DB. Roughly 1% coverage.

What's Next (Offensive)?

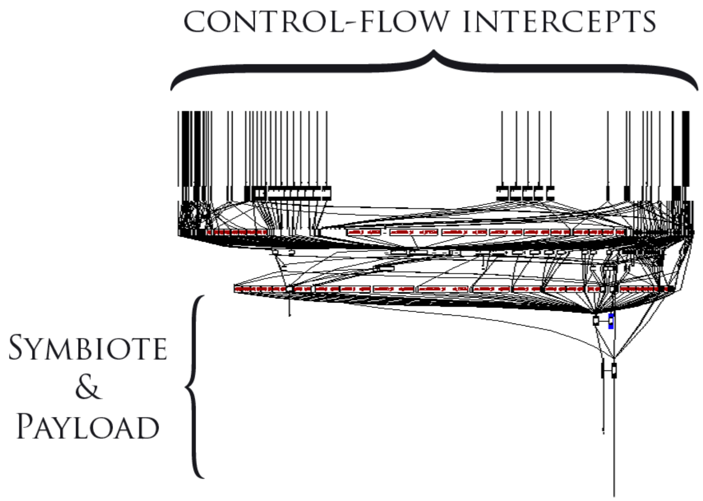
- More comprehensive fingerprint database
  - ~3,000 images in the fingerprint DB. Roughly 1% coverage.
- EEPROM resident malware
  - Current Rootkit will not survive IOS update
  - Better to live in EEPROM
    - Line cards
    - Network modules
    - Motherboard EEPROM

WHAT'S NEXT (OFFENSIVE)?

- MORE COMPREHENSIVE FINGERPRINT DATABASE
    - ~3,000 IMAGES IN THE FINGERPRINT DB. ROUGHLY 1% COVERAGE.
- EEPROM RESIDENT MALWARE
    - CURRENT ROOTKIT WILL NOT SURVIVE IOS UPDATE
    - BETTER TO LIVE IN EEPROM
        - LINE CARDS
        - NETWORK MODULES
        - MOTHERBOARD EEPROM
- LAWFUL INTERCEPT HIJACKING, ROUTING SHENANIGANS, BE CREATIVE!

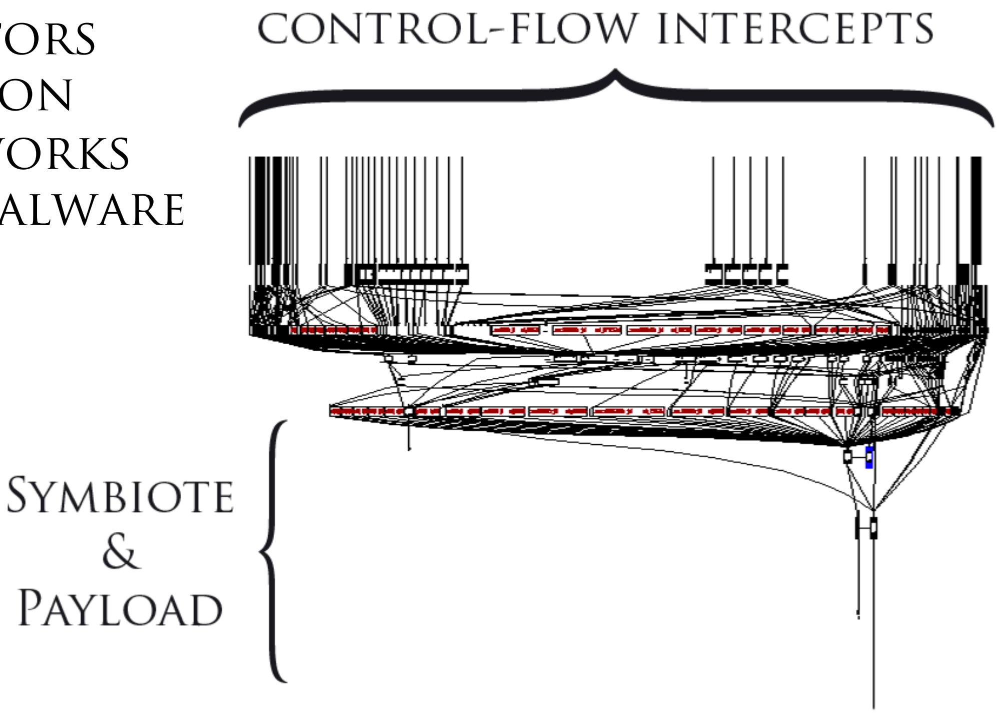
WHAT'S NEXT (DEFENSIVE)?

- SOFTWARE SYMBIOTES
  - GENERIC HOST-BASED DEFENSE FOR EMBEDDED DEVICES.
  - "DEFENDING LEGACY EMBEDDED SYSTEMS WITH SOFTWARE SYMBIOTES"
  - TO APPEAR IN RAID 2011. LOOK OUT!



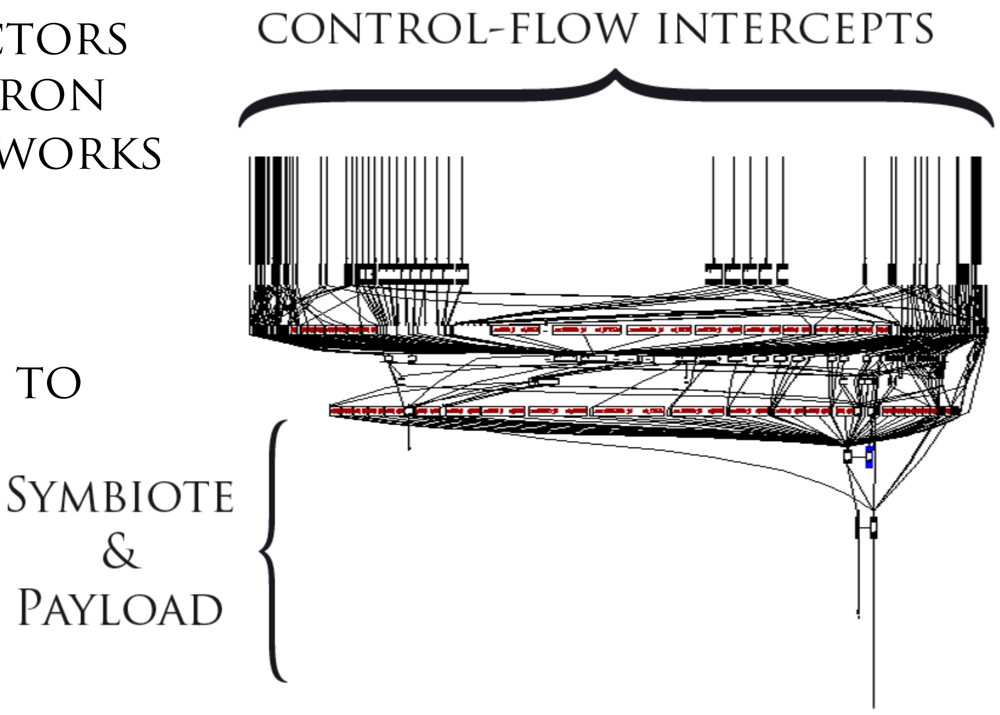
WHAT'S NEXT (DEFENSIVE)?

- CISCO IOS ROOTKIT DETECTORS
    - RUNS ON REAL CISCO IRON
    - DEPLOYED IN REAL NETWORKS
    - WILL CATCH REAL IOS MALWARE



WHAT'S NEXT (DEFENSIVE)?

- CISCO IOS ROOTKIT DETECTORS
    - RUNS ON REAL CISCO IRON
    - DEPLOYED IN REAL NETWORKS
    - WILL CATCH REAL IOS MALWARE
    - A FRIENDLY SHOOTOUT TO TEST OUR DEFENSES? -)
    - PLEASE CONTACT US!



Killing the Myth of Cisco IOS Diversity

Answers!

- Feel free to contact us
  - {ang|sal}@cs.columbia.edu
- Please checkout our publications and ongoing research
  - http://ids.cs.columbia.edu

- This work was partially supported by:
  - DARPA Contract, CRASH Program, SPARCHS, FA8750-10-2-0253
  - Air Force Research labs under agreement number FA8750-09-1-0075

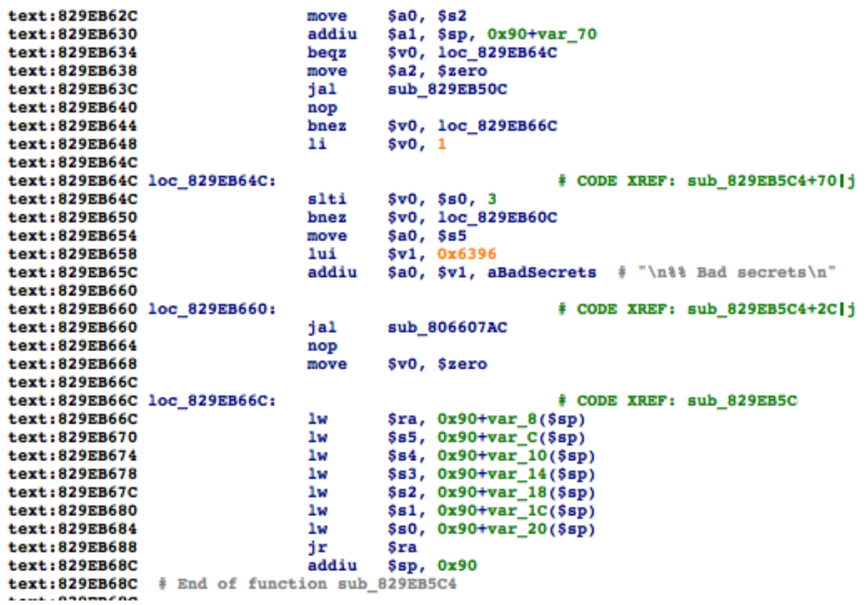
Columbia University
in the City of New York

Killing the Myth of Cisco IOS Diversity

Backup slides

DISASSEMBLING SHELLCODE #1

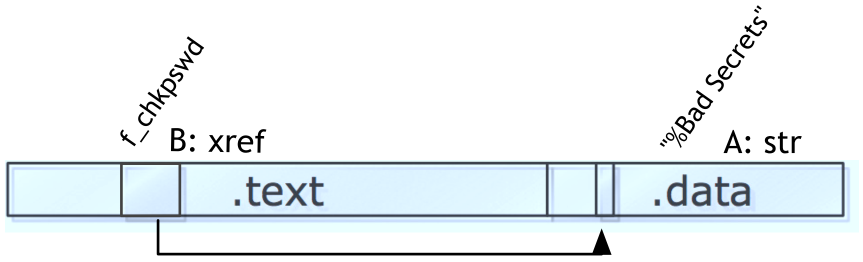
- ORIGINALLY PRESENTED BY FELIX LINDER



```
text:829EB62C                 move    $a0, $s2
text:829EB630                 addiu   $a1, $sp, 0x90+var_70
text:829EB634                 beqz    $v0, loc_829EB64C
text:829EB638                 move    $a2, $zero
text:829EB63C                 jal     sub_829EB50C
text:829EB640                 nop
text:829EB644                 bnez    $v0, loc_829EB66C
text:829EB648                 li      $v0, 1
text:829EB64C
text:829EB64C loc_829EB64C:                            # CODE XREF: sub_829EB5C4+70|j
text:829EB64C                 slti    $v0, $s0, 3
text:829EB650                 bnez    $v0, loc_829EB60C
text:829EB654                 move    $a0, $s5
text:829EB658                 lui     $v1, 0x6396
text:829EB65C                 addiu   $a0, $v1, aBadSecrets  # "\n%% Bad secrets\n"
text:829EB660
text:829EB660 loc_829EB660:                            # CODE XREF: sub_829EB5C4+2C|j
text:829EB660                 jal     sub_806607AC
text:829EB664                 nop
text:829EB668                 move    $v0, $zero
text:829EB66C
text:829EB66C loc_829EB66C:                            # CODE XREF: sub_829EB5C
text:829EB66C                 lw      $ra, 0x90+var_8($sp)
text:829EB670                 lw      $s5, 0x90+var_C($sp)
text:829EB674                 lw      $s4, 0x90+var_10($sp)
text:829EB678                 lw      $s3, 0x90+var_14($sp)
text:829EB67C                 lw      $s2, 0x90+var_18($sp)
text:829EB680                 lw      $s1, 0x90+var_1C($sp)
text:829EB684                 lw      $s0, 0x90+var_20($sp)
text:829EB688                 jr      $ra
text:829EB68C                 addiu   $sp, 0x90
text:829EB68C  # End of function sub_829EB5C4
```

SOMEWHERE IN EVERY IOS IMAGE…

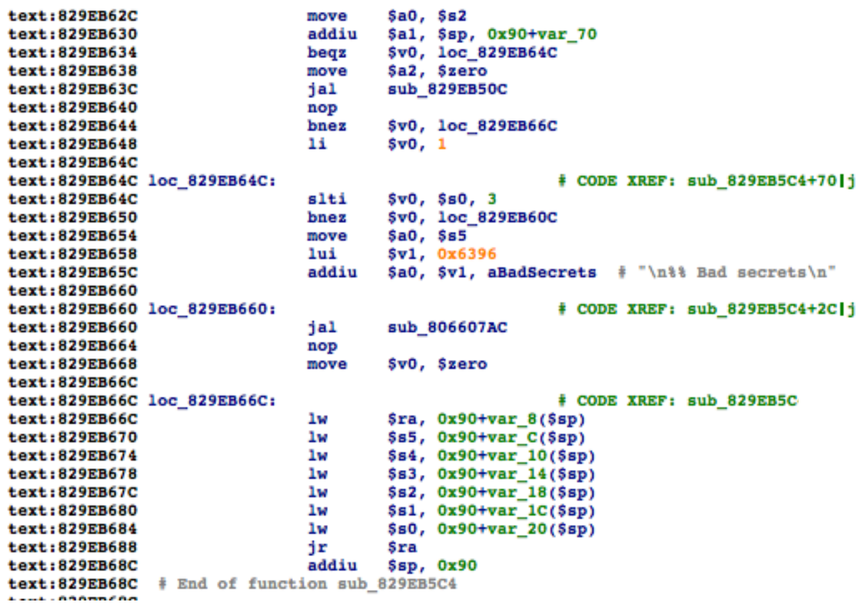
```
FLAG = PASSWORDISRIGHT()

IF (FLAG!=0){
  ROOTME()
}
ELSE {
  PRINTF("BAD SECRETS –(")
}
```



Killing the Myth of Cisco IOS Diversity

Disassembling Shellcode #1

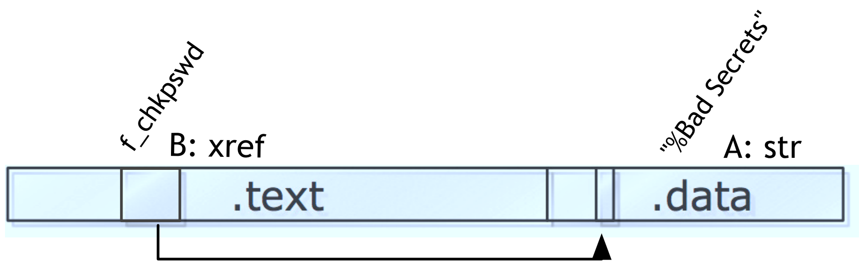
- Originally presented by Felix Linder



```
text:829EB62C                  move    $a0, $s2
text:829EB630                  addiu   $a1, $sp, 0x90+var_70
text:829EB634                  beqz    $v0, loc_829EB64C
text:829EB638                  move    $a2, $zero
text:829EB63C                  jal     sub_829EB50C
text:829EB640                  nop
text:829EB644                  bnez    $v0, loc_829EB66C
text:829EB648                  li      $v0, 1
text:829EB64C
text:829EB64C loc_829EB64C:                            # CODE XREF: sub_829EB5C4+70|j
text:829EB64C                  slti    $v0, $s0, 3
text:829EB650                  bnez    $v0, loc_829EB60C
text:829EB654                  move    $a0, $s5
text:829EB658                  lui     $v1, 0x6396
text:829EB65C                  addiu   $a0, $v1, aBadSecrets  # "\n%% Bad secrets\n"
text:829EB660
text:829EB660 loc_829EB660:                            # CODE XREF: sub_829EB5C4+2C|j
text:829EB660                  jal     sub_806607AC
text:829EB664                  nop
text:829EB668                  move    $v0, $zero
text:829EB66C
text:829EB66C loc_829EB66C:                            # CODE XREF: sub_829EB5C
text:829EB66C                  lw      $ra, 0x90+var_8($sp)
text:829EB670                  lw      $s5, 0x90+var_C($sp)
text:829EB674                  lw      $s4, 0x90+var_10($sp)
text:829EB678                  lw      $s3, 0x90+var_14($sp)
text:829EB67C                  lw      $s2, 0x90+var_18($sp)
text:829EB680                  lw      $s1, 0x90+var_1C($sp)
text:829EB684                  lw      $s0, 0x90+var_20($sp)
text:829EB688                  jr      $ra
text:829EB68C                  addiu   $sp, 0x90
text:829EB68C  # End of function sub_829EB5C4
```

Somewhere in every IOS image...

```
FLAG = 1

IF (FLAG!=0){
  ROOTME()
}
ELSE {
  PRINTF("BAD SECRETS –(")
}
```

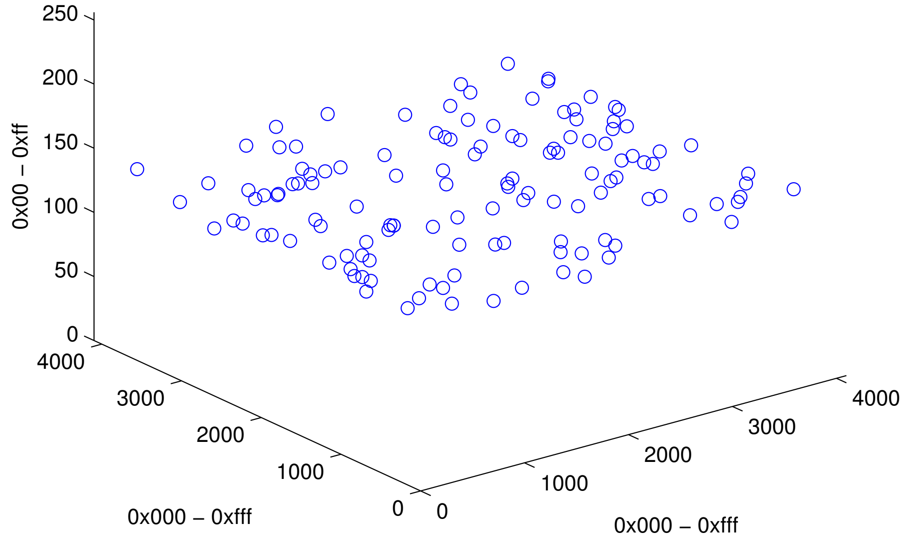


Columbia University
In the City of New York

BlackHat Briefings USA 8.3.2011

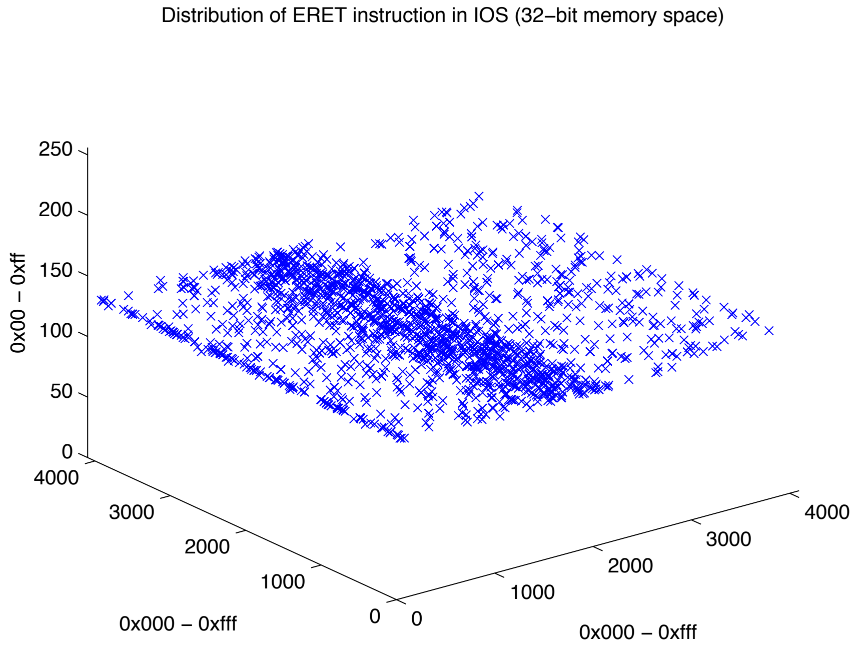
COMPARISON OF POTENTIAL FINGERPRINT FEATURES

Distribution of "Bad Secrets" string x–ref in IOS (32–bit memory space)



- FAIRLY RANDOM, CAN BE USED TO FINGERPRINT IOS
- A SINGLE FEATURE FINGERPRINT
- ONE FIRMWARE, ONE ADDRESS
- POTENTIAL FOR COLLISION HIGHER THAN THE NEXT OPTION

Comparison of potential fingerprint features

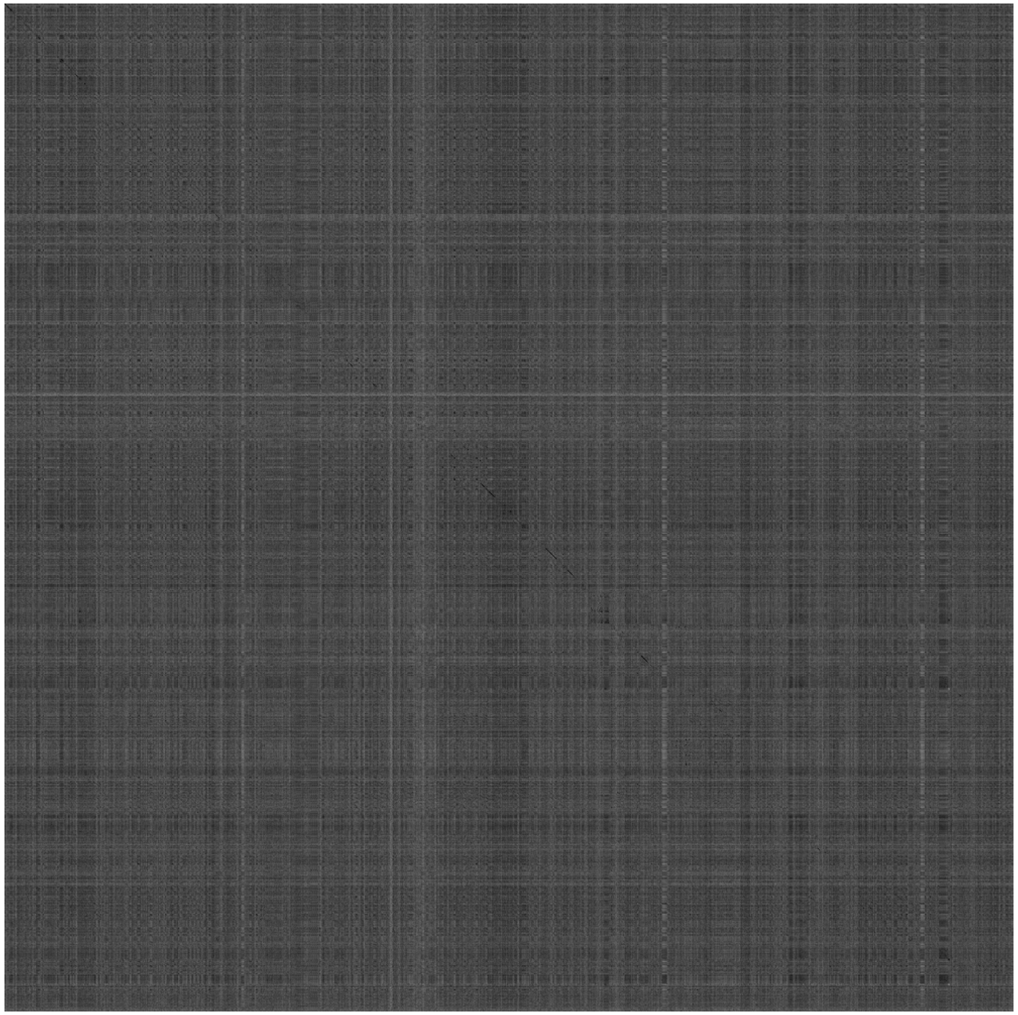


- Concentrated in a predictable range in IOS memory
- Yet diverse enough to uniquely identify unknown firmware version
- Also needed in 2nd stage rootkit, kill 2 birds with one stone
- In our opinion, a pretty good target, but there are many others.
- Multi-vector feature. Each image contains approximately 6-30 ERET instructions.

The basic idea

- Reduce (binary) diverse target to a (functional) monoculture
- Take advantage of offline processing
  - Use a two-phase attack
  - Build a database of device fingerprints
  - Macro-ize 3rd stage payloads, generate device specific payloads on the fly

Killing the Myth of Cisco IOS Diversity



For example

Dotplot of two minor revisions of 12.4 IOS images for the same hardware

IOS 12.4-**23b** vs 12.4-**12**
Cisco 7200 / NPE-200