



Punk Ode Hiding Shellcode in Plain Sight

Greg MacManus Researcher of stuff, iDefense Labs

Michael Sutton Director, iDefense Labs



Where it all comes together:

Contains VeriSign Confidential and Proprietary Information

Agenda

- + Background
 - _ Shellcode history

+ What is Punk Ode?

- _ NOP sled generator++
- Shellcode steganography
- Image encoder with shellcode integrity checking
- + Embedding shellcode in image files
 - _ PNG images
 - _ JPEG images
 - Other Miscellaneous details
- + Demo
 - _ NOP-Palette
 - _ The other tool
- + (Im)practical applications
 - _ Efficient NOP sled generation
 - Bypass active scripting requirement
 - _ IDS evasion

+ Other file formats

- _ Sound files
- _ Video files
- _ Office documents
- _ Text files?



- + In the beginning there were trivial command line or environment variable overflows
- + The First Exploit Code (for a lot of people)

Smashing the stack for fun and profit – Aleph1

- Find a command-line overflow in a setuid root program
- Put a long string, the return address, a NOP sled and the shellcode into an environment variable (EGG) and spawn a shell.
- Run "./vulnerable-program \$EGG"
- Profit!

+ Even trivial exploits were not quite trivial

- _ As everyone knows from the Aleph1 paper, shellcode cannot have null bytes because it would terminate the string.
 - - So the shellcode has to be carefully written to avoid them.
 - Or you could just use Aleph1's code. (Results 1-10 of about 11,800 for Aleph1 shellcode.)



Smashing the Stack – without removing null bytes

main(int argc,char *argv[])

```
char *env[32]; int i=0;
if(getenv("XXX"))abort();
env[i++]="XXX=";
env[i++]="\xC7\xC0\x01";
env[i++]="";
env[i++]="";
env[i++]="\xC7\xC3\x01";
env[i++]="\xcd\x80";
env[i++]=0;
execve(argv[0],argv,env);
```

(gdb) r

Starting program: /tmp/x

Reading symbols from shared object read from target memory...done.

Loaded system supplied DSO at 0xb7f76000

```
Program received signal SIGABRT, Aborted.
```

```
0xb7f46402 in ?? ()
```

(gdb) x/3i environ[1]

0xbfa45fe6: mov \$0x1,%eax

0xbfa45fec: mov \$0x1,%ebx

0xbfa45ff2: int \$0x80

(gdb) # so it looks like you can store shellcode with nulls in the environment.



 (\star)

Shellcode History – cont'd

- + The previous trick is probably not all that useful now.
 - _ Non-executable stack and ASLR are becoming more common.
- + Some other limitations of using the environment or command line
 - _ Remote vulnerabilities
 - Difficult to get the remote server to start with \$EGG in the environment
 - But there are CGI programs
 - Limited buffer sizes
 - How small can you make your shellcode?
 - _ Not just null bytes to worry about.
 - More restricted characters as you look at more targets.
 - No newlines, no spaces, no tabs, no commas, no colons, no semi-colons, no equals, no high bits, no control characters at all.
 - Write special shellcodes by hand for each special case!
 - Hard work

IDS evasion

 As more people learned about the techniques, people started wanting to know how they were getting owned.



- + Shellcode now is much more advanced than executing /bin/sh
 - E.g. Metasploit 2.5 contains 25 different payloads and 9 encoders for Win32, from simply executing a command to injecting a VNC server into a remote process.
 - According to the schedule HD is actually talking in another room about the latest version of Metasploit right now. And you are missing it.
- + Automatic shellcode generation.
 - Most use decoder stubs which extract and call the actual code
 - XOR encoding
 - Alphanumeric shellcode
 - Unicode shellcode
 - NOP-sled generators
 - to avoid IDS NOP-detection.
 - If you see 1000 0x90 characters in a row go to your server, it may be part of an exploit.



What is Punk Ode?

- + Punk Ode = Pun Code
 - _ pun (n)
 - 1. A play on words, sometimes on different senses of the same word and sometimes on the similar sense or sound of different words.
 - _ Exploits the von Neumann architecture
 - Data and code are indistinguishable to the computer
 - The meaning of a sequence of bytes is dependant on context
 - If you throw them at the screen, they become pixels
 - If you make the computer jump into them, they become code

```
$ cat hello-von-Neumann.c
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello, von Neumann architecture!\n");
}
$ cc -o hello-von-Neumann hello-von-Neumann.c
$
```



The hello-von-Neumann.exe binary embedded in an image.



+ Pun code

- _ Punk Ode (n)
 - A play on bytes.
- _ It makes images which are visual puns
 - Entirely comprised of a NOP-sled and shellcode.
 - Load directly into your web browser
 - PNG, GIF or JPEG format
 - They look pretty much like lormal pictures.
 - Only slightly(*) more work than embedding the shellcode in a large image file to get it in memory
 - Not just a theory.
 - Actual working exploits have been tested using this method.



Proto-punk Ode – Way back in '02

- First stage of research was many years ago
 - Making huge, compressed NOP-sleds.
- Owning Netscape 6 for Fun and Profit
 - _ Netscape Bug Bounty
 - \$US1000 for code execution bugs.
 - This motivated the initial research.
 - _ An early attempt
 - The colored stripes are shellcode, other parts are "\xeb\x40" repeated.
 - If you saw this you would know 'something is odd'.

A section of the NOP-sled/shellcode file from an exploit for the 'jar' bug reported here: <u>http://bugzilla.mozilla.org/show_bug.cgi?id=157646</u>



Proto-punk Ode – The original motivation

- + The main question the initial version was written to answer was:
 - "How can I find my shellcode?"
 - The internal state of a web browser could be pretty much anything at the time the victim was exploited.
 - Multiple things going on
 - Handling user input, Drawing the interface, Downloading images, webpages, Updating cache files, Executing JavaScript, helper applications, Creating and destroying objects, Checking mail and newsgroups.
 - The heap was in an unknown state.
 - How long ago did they start the program? How many windows are open?
 - If only there was a way to make the program allocate a huge amount in a fixed location...
 - *malloc(size)* used an *mmap()*ed chunk over certain size value.
 - for large enough values, it was almost always in a predictable range
 - _ The only thing left was working out how to put shellcode in that range.



Exploits the way people look at things

- The eye is very forgiving of certain types of display errors.
 - Relative brightness is more important than absolute.
 - Slight change in a color over space may be ok.
 - A little 'noise' may be ok.
 - Certain colors can be changed more than others.
 - Brighter colors can have more variation from the original values and still look ok.



How are images stored?

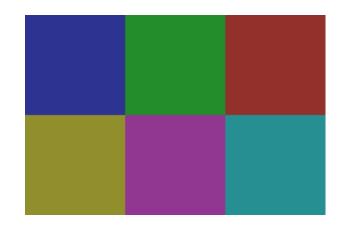
- + 2D image = 1D array of bytes
- + Color encoding
 - _ Grayscale
 - _ RGB
 - _ CMYK
 - _ YCrCb
 - _ Indexed
- + Compression
 - _ None e.g. Raw BMP
 - Lossless e.g. PNG (zlib deflate compression, like gzip/pkzip)
 - Lossy e.g. JPEG (psychophysics using discreet cosine transfer)
- + File in memory
 - _ Decompression results in 1D array in memory



Punk Ode – PNM

- + PNM (Netpbm superformat)
 - _ PBM Portable Bitmap
 - 1 bit per pixel
 - _ PGM Portable Graymap
 - 8 bit per pixel (grayscale)
 - PPM Portable Pix Map
 - 24 bit per pixel
- + Very simple image format
- + Easy to process and generate
 - Used as intermediate format for Punk Ode
- + Many open source utilities
 - _ Netpbm () netpbm.sourceforge.net/
- + Punk Ode
 - Makes extensive use of this format.

Contents of a PPM format image file The 4th line is the image data.



How the above PPM format image file displays (enlarged from 3 pixels by 2 pixels)



_ A PNG contains at least

- A PNG header
- An image header,
- A image array (compressed with the gzip 'deflate' method)
 Each row has an extra byte which does stuff to make the image smaller.
- An image footer.
- Every header (apart from the PNG header) has its CRC-32 stored next to it.
- + Additional relevant features supported by the specification and some viewers.
 - Color correction mappings
 - gAMA, cHRM, sRGB, iCCP
 - _ Transparency/Alpha
 - Some programs store 24 bit pixels padded to 32-bit per pixel.
 - The extra byte is sometimes for the alpha value.
 - If not, you could try writing shellcode which has every 4th byte set to the padding value.



Punk Ode – PNG

+ Example

(*)

- _ Original image GIF
 - 9KB
- Converted to PNM, scaled up,
 - 115MB
- _ Passed through NOP-Palette to add shellcode and converted to PNG
 - 1.2 MB



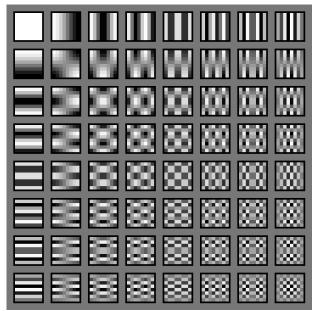


- + More complex
 - _ JFIF JPEG Format Image File
 - _ Joint Photographic Experts Group (JPEG)
 - Designed for photographs
 - Not appropriate for images with sharp lines and high contrast areas
 - _ Lossy
 - Psychophysics Removes detail typically unnoticeable by human eye
 - Adds complexity for Punk Ode will mangle shellcode and NOPs
 - Changing a value in one location can impact adjacent locations
- + Punk Ode
 - _ Restricted values used for NOP sled and shellcode
 - _ 'Lossy' shellcode
 - Maximize JPEG quality to limit loss
 - Shellcode with padding that can be removed without major impact
 - Carefully align shellcode
 - We have a decoder stub which is 'JPEG safe'



Punk Ode – JPEG

- + Discrete Cosine Transfer (DCT)
 - _ Breaks image into 8x8 pixel blocks
 - _ Converts each block from spatial domain to frequency domain
 - Discards those frequencies which don't contribute 'much' to the overall image
 - _ Subject to rounding errors, therefore even maximum quality JPEG
 - images will experience loss





Punk Ode – Windows Byte Order

- In memory, Microsoft often stores images in BGR (blue, green, red) byte order instead of RGB, which is utilized by most image formats
- + Punk Ode must account for this by flipping the red and blue components of the shellcode and multi-byte NOPs
- + Internet Explorer handles images this way
 - _ This adjustment is trivial for simple formats such as PNM and PNG
 - _ This is a significant adjustment for complex formats such as JPEG
 - Separate JPEG decoder stub created to handle this



Punk Ode – Color mapped Files

- + Indexed file format (e.g. GIF)
 - Byte array stores pointers to a color map
 - _ Limits the number of colors that are available
- + Punk Ode
 - Limited color map also restricts values that can be used for shellcode and NOPs
 - By changing the order of the color map, you can change the contents of the file when compressed without impacting the structure of the image when uncompressed
 - This provides a trivial IDS evasion technique
 - Similar to Metasploit's WMF httpd using gzip transfer encoding.



What about vector based images?

- + Vector based image formats
 - _ For example: SVG, WMF, CDR
 - _ Based on lines, curves and filled areas instead of dots
 - _ Almost everything described in this talk *could* work with vector graphics.
 - Vector graphs aren't as widely supported as raster images currently are.
 - MS06-001 WMF AbortProc() exploitation used a helper application.
 - Depending on the type of file and the viewing application, it may not even be rendered to memory which is accessible
 - EMF and WMF for example do not appear to be rendered to a memory buffer in the Windows Picture and Fax Viewer, but instead seem to only use the GDI interface to handle display.
 - Shellcode could still be put inside a file, crafted in such a way as to still render correctly.
 - could be put into an embedded image file within WMF file.



Punk Ode – How the tools relate to the process

- + Punk Ode is a collection of script, programs and methods
 - _ Start with PNM base image
 - This is what we want our shellcode to resemble
 - Convert to NOPs and add the shellcode
 - The tool NOP-Palette
 - No promises, but better looking images at the cost of extra processing is the in development.
 - The Other Tool
 - This is for JPEG tricks.
 - _ Convert to Final Image type
 - PNG
 - Netpbm tool 'pnmtopng'
 - If you're good, we may integrate some other code to NOP-Palette to do this.
 - JPEG
 - libjpeg tool cjpeg with special command line options



+ NOP-Palette

+ The Other Tool



*

- 1. Select original image
- 2. Rescale image
 - + Larger images take up more memory
- 3. Convert image to NOP sled
 - + Map each byte of the image to a valid op-codes ('mostly' NOPs)
- 4. Embed shellcode
 - + Preferably somewhere it is not obvious (e.g. border, noisy area, etc.)
- 5. Convert this file to the final format
 - + Lossy compression formats and other image traits may modify code
 - + These modifications can often be accounted for



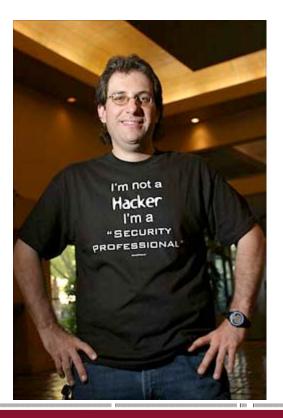
- + The purpose of the NOP-sled is to give the CPU something to do while it's on its way to the shellcode.
- + Use safe instructions
 - never changes important stuff.
 - 0x90 xchg EAX, EAX
 - More commonly known as 'NOP'.
 - 0x42 inc EDX
 - So long as you don't need the original value of EDX in the shellcode.
- + Unsafe instructions
 - these will usually crash the sled.
 - push, pop, call, ret
 - all require a usable stack
 - Anything which dereferences a register
 - Privileged operations

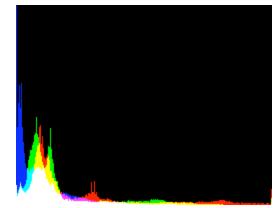




Punk Ode – Byte Frequency Counts

- + The original image I think he's a hacker or something.
 - _ The byte counts show the relative amounts of each color component (the Y-axis) by the intensity of that color component (the X-axis).



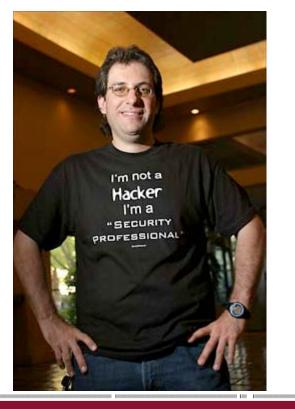


Intensity histogram of original image

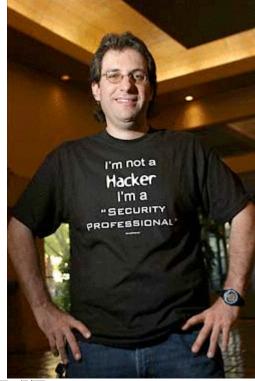


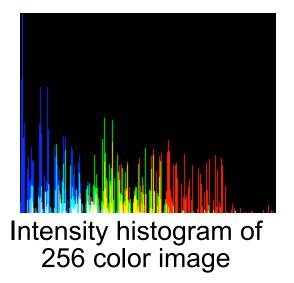
Punk Ode – Spot the difference

- + Comparison of original and 8-bit indexed image
 - _ Nothing extremely obvious has changed.
 - _ The histogram has spikes.



256 color optimal palette.







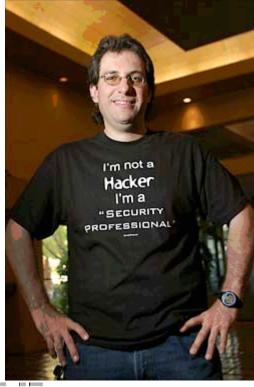
 (\star)

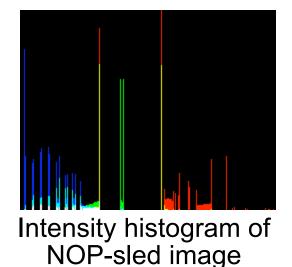
Good enough to pass casual inspection?

- + Comparison of original and NOP encoded images
 - _ His right arm looks like he's been dumpster diving somewhere nasty.
 - The original color values were in a range which contained lots of 'unsafe' instructions, and the closest safe ones were chosen.



Whole image is a NOP-sled





The nearly empty space in the center contains a large number of 'unsafe' instructions.



- + The previous image was just a NOP-sled
 - We pretty much just changed the palette the image used.
 - Also stopped it using too many prefixes and a couple of other things.
 - _ It's possible to be more clever making the sled
 - The range of op-codes from 0x70 to 0x7f is in the 'unsafe' range
 - These are jump short instructions, so they could be safe under some conditions.
 - Other tricks based on what you are doing
 - Only converting parts of an image
 - From offsets in the image, and memory
 - Analyzing it as you go may let you
 - Force some conditional jumps and not change the skipped bytes.
 - Not modify opcode arguments
 - A few other tricks.
- + Shellcode constraints
 - Lossless vs. lossy image compression
 - What you see is not always what you got.
 - Lossy compression kills shellcode.
 - You have to code around it.



+ Challenge

- Lossy compression results in loss of information, this can damage instructions (NOPs, and shellcode)
 - damage can be is non-localized
 - Available instructions may be dependent upon instructions several bytes away, or several lines away.
 - Solutions

1. Carefully hand coded shellcode

- Can be done using multiple iterative checks of code which is time consuming.
- Must adjust shellcode or placement if:
 - Output image ≠ Input image
 - Shellcode altered
 - Semi-automatic modification is possible, at least for JPEG...
- 2. Shellcode decoder stub
 - Stub decoder uses the 1st approach so that the main shellcode doesn't have to.



- + Sound
 - _ PCM encoded WAV files just like PNM except with sound
 - _ Other lossless types are somewhere between PNG and GIF
 - _ MP3 More like JPEG
- + Video
 - MPEG,AVI
 - Most are like JPEG, but harder to understand.
 - Lossless types also exist RLE encoding for example
- + Differences from images
 - _ These aren't usually decompressed into memory all at once.
 - Bad perhaps we can't fill as much memory at once
 - Maybe we can... how many frames does it buffer?
 - Good We may not need to have a valid file if its big enough
 - This isn't really Punk Ode style though, just using big files.



Punk Ode – Other file formats

+ Office Documents

- _ Can store PNG files in them
- _ Remember, if you can't see it, it's not Punk Ode
 - But it still may work.

+ PDF

- _ Works great.
- _ Easy to inject the code.

+ Text files

- _ Wouldn't it be nice to be able use alphanumeric shellcode which read
 - like something in a natural language written by a human?
- _ Is it possible?



Punk Ode - (Im)practical applications

- + Why use a big image file?
 - When you can't control the return address, make the return address valid.
 - Doesn't require JavaScript like variations on SkyLined's heap spraying technique.
 - His method fills the heap with a long run of 0x0d followed by the shellcode.
 - The 0x0d bytes mean 2 (or 3) things, depending on how they are being used
 - A 5 byte NOP-like instruction if it
 - OR EAX, 0d0d0d0d
 - A valid address (which points to a valid address)
 - 0d0d0d0d: 0d0d0d0d ; self-referential pointer
 - Many workarounds for vulnerabilities only workaround the exploit.
 - Disabling JavaScript only stops JavaScript based vulnerabilities
 - Doesn't prevent weird HTML or other file format parsing exploits
 - It could prevent an exploit using SkyLined's shellcode method
 - Would not stop Punk Ode method.



- + Look for big files!
 - Compression is good
 - 8192x8192 light gray image with shellcode.
 - 192MB uncompressed raw image
 - PNG: 83KB, using mod_gzip to transfer it: 1084 bytes

+ Byte Frequency graphs

- The histograms for the normal images and the NOP-sled image in the examples were totally different.
 - Page aligned memory allocation lets you do tricks with relative offsets in the file so, without knowing precisely where in the image you are going to land, you can modify only those areas where are at the right offset, modulus the page size.
 - The rest of the image can be crafted to cancel out any disturbance from the expected frequencies.

Possible counter-counter measure

- Slice up the image and analyze them slices independantly
- Look for variations in the variations between slices from different regions.
- A lot of work. Definitely not something to do realtime.



IDS would have to be very fast and very smart

- What the targeted program is.
- How the targeted program handles that image format
 - E.g: gAMA chunks in PNG format images: controls how things brighten as the value increases.
 - What is the default gamma value for the target application?
 - Will it even read it from the file?
 - What if the file is slightly malformed?
 - 2 gAMA tags?
 - A gAMA tag after the IDAT?
 - Does it do the rightit correctly?
 - filter:progid:DXImageTransform.Microsoft.AlphaImageLoader()?!
 - Where does the compositing happen?
- How many files which expand to over 100MB of what could be interpreted as shellcode can your IDS system analyze per second?

If the IDS can emulates the browser, can the IDS be owned at the same time? ;]



Speculation on detection

- + It may be possible to detect abnormal compression patterns
 - Compare with compression applied to determine authorship/language of text files.

+ But you really should be looking for some vulnerability being triggered, not shellcode.



One final nop sled example.

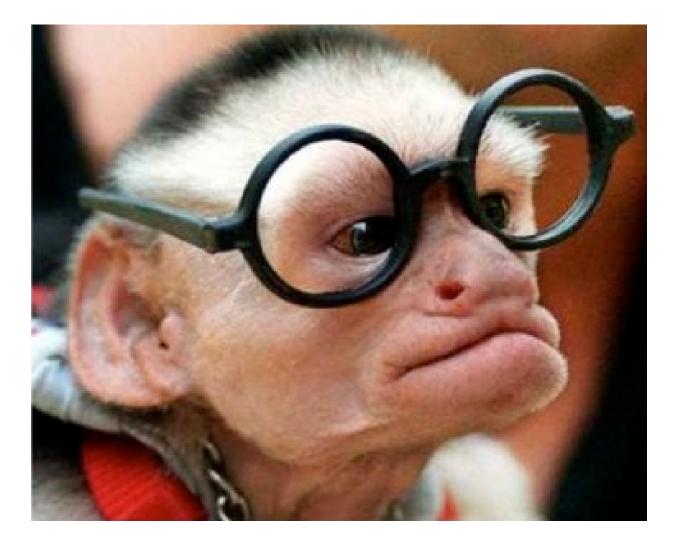




*

Questions?

*-





Contains VeriSign Confidential and Proprietary Information