DigSig novelties

Libre Software Meeting 2005 – Security Topic
July 2005

DigSig Team:
Axelle Apvrille, David Gordon, Serge Hallyn,
Makan Pourzandi, Vincent Roy

Outline

- Introduction: why and what for ?
- How: mechanisms involved in DigSig
- Short example: DigSig in use
- Recent Developments (2005)
- Future work

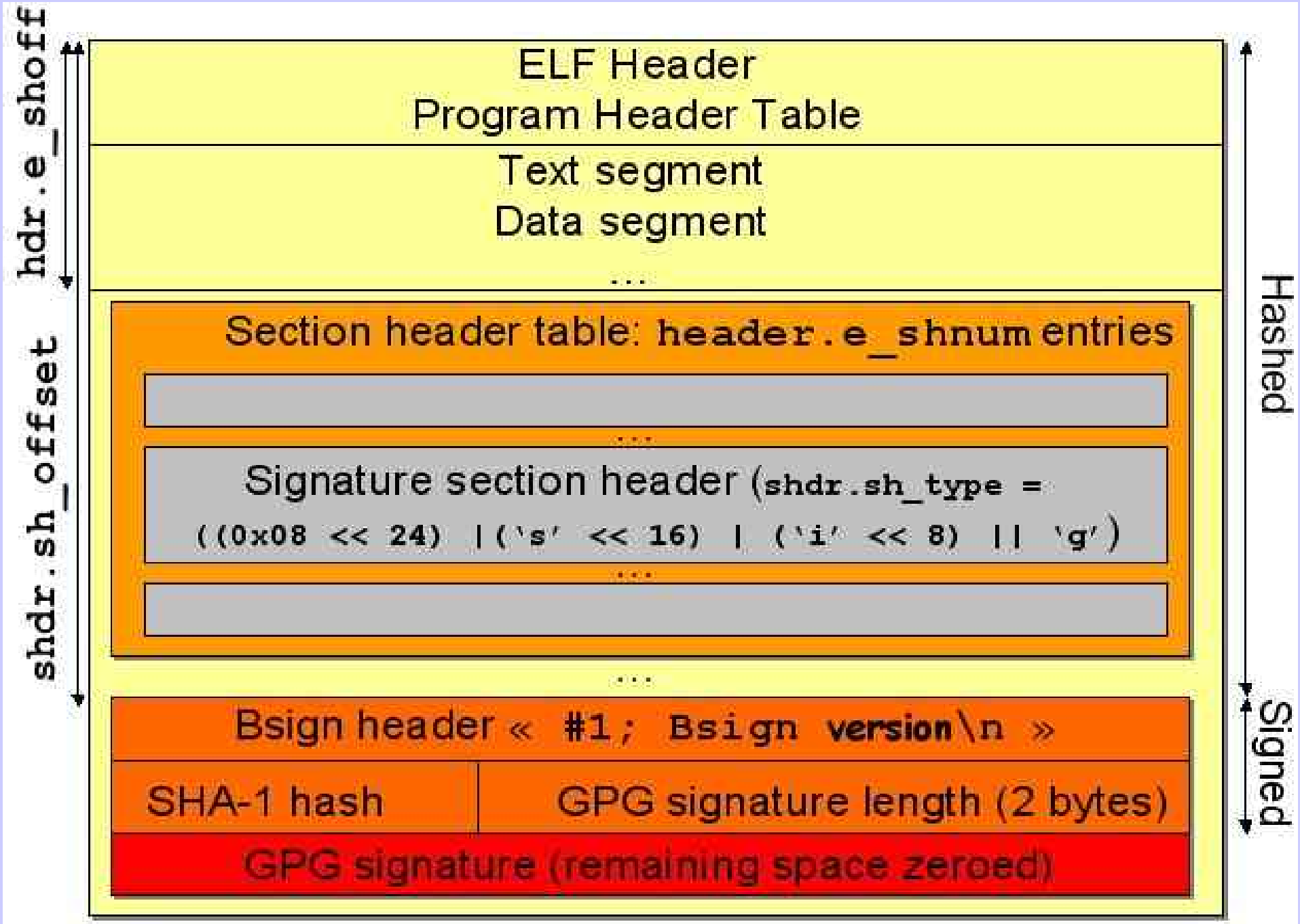
DigSig: why and what for ?

- Why ?
  - Increasing impact of of malware (virus, worms) – mainly on Windows, but also on Unix
  - Users are often careless (email attachments, download of Trojaned soft...)
  - Once compromised, attacker hides its activity
  - Firewalls aren't sufficient
- What for ?
  - Prevent execution of malware
  - Does not prohibit malware download
  - Only for ELF binaries, on Linux.

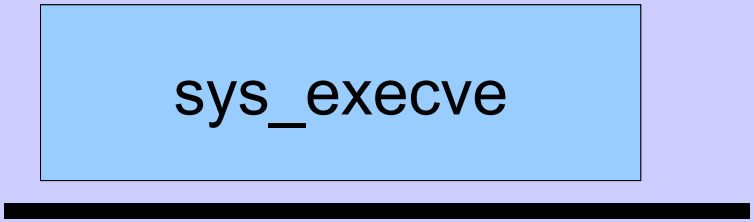
The DigSig Solution

- Preliminary steps: get an RSA key pair
- Embed a signature within ELF binaries and libraries
  - No need to keep a signature database
  - It's already done: we use Bsign
- At run-time,
  - The Linux kernel automatically verifies the signature. The binary only gets to run if signature is okay.
  - No special command to launch
  - Internal security is enhanced

Bsign embedded signature



Example: binary execution



do_execve

bprm_alloc_security: nothing to do, no context

search_binary_handler

bprm_check_security: nothing to do (except scripts)
libraries don't get into this hook

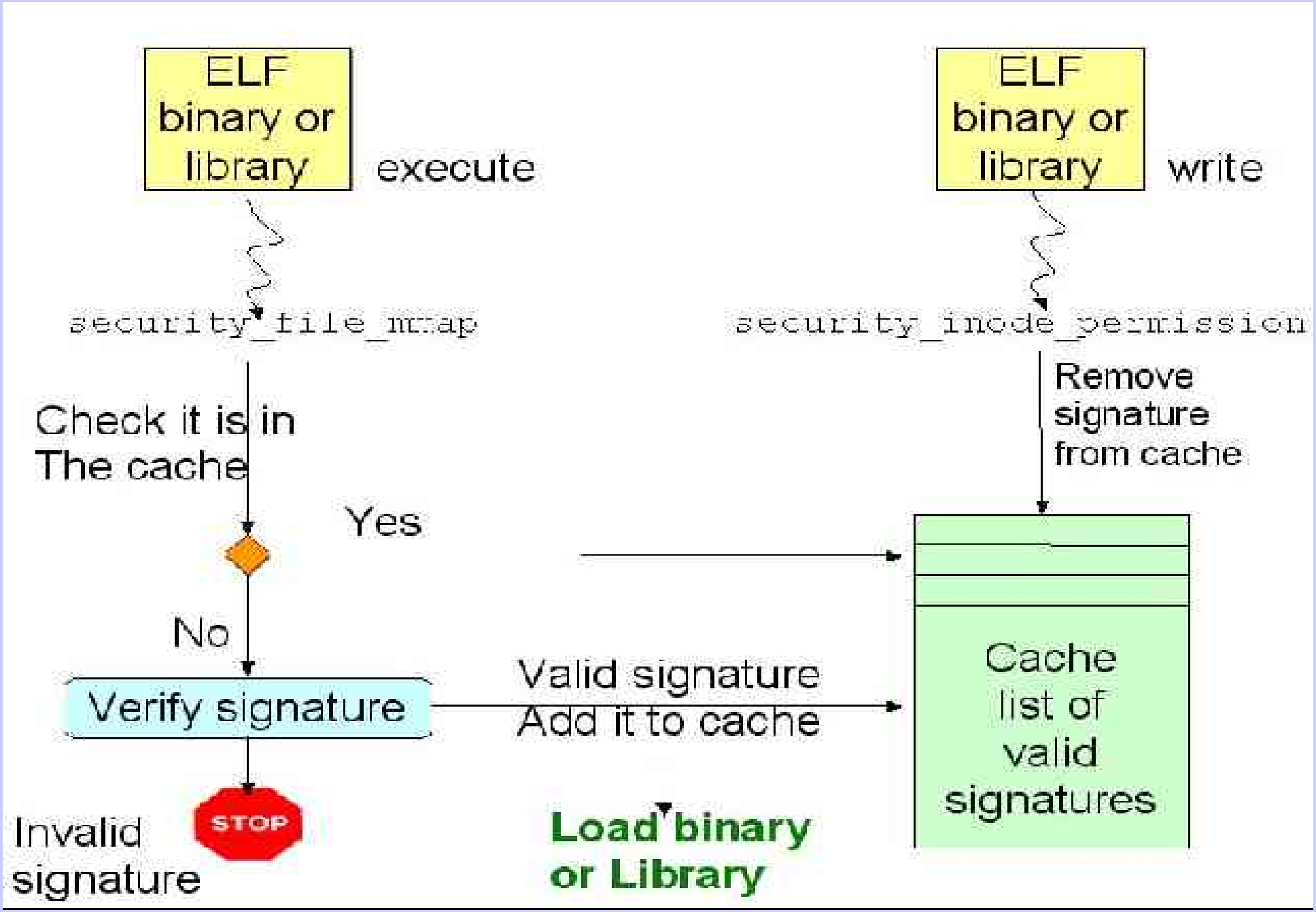
load_elf_binary

do_mmap

file_mmap: load & verify signature

bprm_free_security: no security context

Signature caching



Signature Revocation

Administrator signs application A: SIG(A)

A's security at stake: administrator
revokes SIG(A)
gets a newer version, or another soft (more secure)
signs the soft: SIG(A')

A is compromised

A cannot run, A' can

<u>Revocation details:</u>

- provide revocation list at startup
- Retrieve signatures: ./ tools/extract_sig malicious.bin sig
- Insmod -f ./digsig_verif.ko
- For each revoked signature: cat file.txt > /sys/digsig/revoke
- only root can write there

Hands on DigSig (1)

- Download Digsig v1.4.1 (or +): http://disec.sourceforge.net
- Requirements: gpg (1.2.2+), bsign v0.4.5, 2.6.8 (or +) kernel with
  - CONFIG_SECURITY=Y
  - CONFIG_SHA1=Y
- Compile
  - DigSig.init utility:
  - ./digsig.init compile
    - This actually calls make -C /lib/modules/`uname -r`/build SUBDIRS=$PWD modules

Hands On ...(2)

- Generate key pair
  - Gpg –gen-key ...
- Extract public key
  - gpg –export –homedir=... >> key.pub
- Sign binaries and libraries
  - bsign -s -v -l -i / -e /proc -e /dev -e /boot -e /usr/X11R6/lib/modules
- Secure the private key
- Make sure only root access for /sys/digsig

Hands On ... (3)

- Start digsig:
  - ./digsig.init start key.pub
  - Loads the kernel module
    - Option: signature cache size
      - `insmod -f digsig_verif.ko digsig_max_cached_sigs=1024`
  - Sets the public key (/sys/digsig/key)
  - Sets the revocation file (/sys/digsig/revoke)
- Try it:
  - ./ps-signed
  - ./ps-unsigned:cannot execute binary file
  - Realistic try: take off the DIGSIG_DEBUG flag in the Makefile, re-compile.

DigSig Performance

Small overhead (~1%) shows during system time

Kernel without DigSig	
real	sys
19m21.890s	1m27.992s
19m9.276s	1m26.584s
19m9.464s	1m26.191s
19m7.717s	1m25.799s
Kernel with DigSig	
real	sys
19m19.957s	1m28.541s
19m7.485s	1m26.832s
19m7.883s	1m26.549s
19m6.494s	1m26.618s

2.6.4 kernel make on Pentium 4 2.4Ghz

The caching mechanism improves performance

Kernel without DigSig	
real	0m0.004s
user	0m0.000s
sys	0m0.001s
DigSig without caching	
real	0m0.041s
user	0m0.000s
sys	0m0.038s
DigSig with caching	
real	0m0.004s
user	0m0.000s
sys	0m0.002s

Time ls -Al on Pentium 4 2.4 Ghz

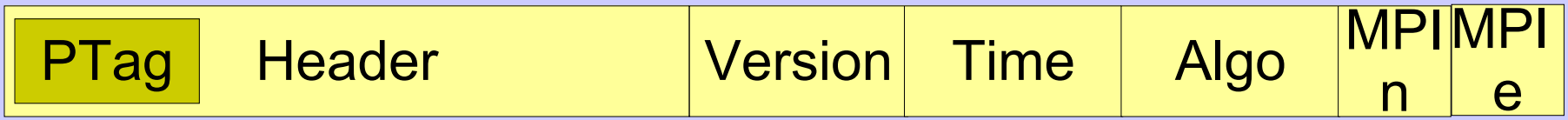
- 1st load overhead slowly grows with executable size (~0.0016 µs / byte), but a standard Debian has no more than 1.8% executables and libraries above 512KB.

Recent developments (1)

- 64-bit architecture support
- Why ?
  - Get 64-bit ELF binaries checked
  - Ensure 32-bit binaries run on 64-bit architectures
- How ?
  - Switch to elf32_* or elf64_* structures depending on the ELF magic number in the ELF header: ELFCLASS32(1), ELFCLASS64(2)
  - Use the appropriate structures and sizes
- Lead: Serge Hallyn

Recent developments (2)

- RSA 2048 support
  - Why ? 1024-bit starting to get slightly « weak », more and more users with 2048-bit keys
- How ?
  - Already okay in the kernel module
  - Update the key extraction tool: read an OpenPGP Public Key packet (6)
- Lead: Axelle Apvrille



Recents developments (3)

- Script support
  - Compute a detached signature (gpg -sb ...)
  - Add signature as an extended attribute (setxattr)
- Run-time kernel signature verification
  - In bprm_check_security
- Problem
  - Won't secure « sh myscript.sh »
- Lead: Serge Hallyn (dev) ... and all team

Recent dev: passwords for module unloading (4)

- Why ? Prevent unsollicited module unloading...
- How ? Require a password for unloading
  - Unless CONFIG_FORCE_MODULE_UNLOAD
- Set password at load time
  - Communicate password via /sys/digsig/passwd
  - digest password (sha1 module)
- Security view: this is only a workaround. Cannot solve the roots of the problem.
- Lead: Marco Slaviero

Recent developments (5)

- LTP (Linux Test Project): ltp.sf.net
- Why ? Critical regression introduced in Digsig v1.3.x, v1.4.0: fixed in v1.4.1
- How ?
  - Standalone scripts in ltp/testcases/kernel/security/digsig
  - Retrieve digsig and put it in ./digsig-latest
  - Make all && sh test.sh
- What does it test ?
  - Impossible to write after execute
  - Impossible to execute after write
  - Try to modify bytes of an executable
- Lead: Serge Hallyn

Recent Issue

- Test
  - Take a signed executable, modify byte per byte, check signature
  - => fact: there are ***a few spots where binary can be modified without detection***
- Locations
  - OpenPGP signature packets v3
    - All bytes are not signed (e.g possible to change signature packet header) => solved in v4
    - Left 16 bits of signed data hash => only intended for quick tests
  - Bsign: zeroized area in signature section

OpenPGP Signature Message

0	1	2	3	4	5	6	7
Ptag	Length (length depends on PTag)		Version (0x03)	Length (0x05)	Type (0x00)	Creation time (4 bytes)	
	Signer Key ID (8 bytes)						
		Public Key Algo	Hash Algo	Left 16 bits H(data + type + creation time)		MPI length in bits	
MPI (signature value)							

Packet header: old format, indicates a signature packet
Signature packet type: 0x00 = signature of a binary document
Public Key Algo: 0x01 = RSA (encrypt or sign)
Hash algo: 0x02 = SHA-1
Signed data = Bsign signature section + type + creation time

Exploiting the Zeroized Area

- (Very) simple «exploit»:
  - Hiding unsigned data in the signed executable
    - Locate the « bsign » signature section
    - Go the end, go back until != 0x00
    - Write data in there
  - ./hide <bsigned-exec> <data-to-hide>

Hiding data in binary (2)

```
[axelle@localhost hidedata]$ ~axelle/softs/bsign-0.4.5/bsign -s hw

Enter pass phrase:
[axelle@localhost hidedata]$ ~axelle/softs/bsign-0.4.5/bsign -V hw
bsign: good signature found in 'hw'.
[axelle@localhost hidedata]$ cat > text.txt
something to hide in the signed binary

[axelle@localhost hidedata]$ ./hide hw text.txt
e_shoff=3414 e_shnum=34 e_shentsize=40
file to hide: size=39
zero area: offset=7983 len=193
Amount read=39 Amount written=39
[axelle@localhost hidedata]$ ~axelle/softs/bsign-0.4.5/bsign -V hw
bsign: good signature found in 'hw'.
[axelle@localhost hidedata]$ tail -n 2 hw
©!ª-ÀãGBîã§k{*@yè B¾Y}b¢fv^£È£þ)|½¾8ÑoS à9ÃÀ¤æ8èE¸Ó!GWgU5±oÑ`ìP×çè. òvZêt¤3ÞW¿õ¹
á¨jqäUÖÙÎ@)ä=ÉÙ¦v'{cØah¿+\çR³Ûa>ù2»     ¦)ci«oH÷ÙLJk5Ñ®aùÿrÎn²d¢þýQì-(&q1_³=@uãm
<^+à>ù  £2ÜÏlÜ[\¦L»÷ÎþvÉ9ïÈHwqã"p[ ÐÚ²pVurÇY
                                        ¹mÊèRòIünlõà;-þßß*fx#¬ÄëíÑùsomething
 to hide in the signed binary
[axelle@localhost hidedata]$
```

Future developments

- Handling script signature in all cases
- Use standard Makefiles instead of digsig.init
- Perhaps enhance crypto with ASM code ?
- Sign the signature revocation directory (tar & sign)
- Securing the public key (integrity)
  - The public key shouldn't be replaced
  - Use a TPM:
  - Bind a public key with platform configuration
    - Create key pair, store key pair in TSS object
    - Create a PCR object, add values to be hashed
    - Wrap the object with a TPM key
    - Check integrity at boot time (automatic)

Related Work

	Run-time sig. verif	File type	Availability
Bsign	No	Binaries	GPL
Cryptomark	Yes	Binaries	Abandonned ?
Signed Exec	Yes	Binaries + scripts	Not GPL
Tripwire	No	All	GPL & commercial
Umbrella	Uses DigSig	Uses DigSig	GPL
DigSig	Yes	Binaries	GPL

DigSig Stacking

- Three possibilities
  - Modify digsig to allow for module stacking
  - Modify the other module to allow for stacking
  - Use the stacker patches at sourceforge.net/projects/lsm-stacker