Netpgp and Signed Execution

Alistair Crooks
The NetBSD Foundation
agc@NetBSD.org
d415 9deb 336d e4cc cdfa 00cd 1b68 dcfc c059 6823
September 2012

Abstract

We have seen various types of digital signature being used to verify the provenance of data for a long time - PGP signatures being used to show that the signed data can be trusted, ssh signing random data to prove that the private key is known. This paper outlines the steps we used to embed digital signature verification into the kernel, and to use this functionality to verify a signature on a binary program, shared library, or interpreted script. The net effect of this is to allow or deny execution of software on a machine, based on a digital signature on the software itself. There are significant benefits to running approved software based on its provenance, especially in the cloud, or on edge machines. Because the signature assures the identity of the software provider (unless the keys are exposed), software may be downloaded at a later date and run on the same machine, again secure in the knowledge of its provenance.

1. Introduction

1.1 The Veriexec Framework

NetBSD has had an in-kernel verified execution facility since 2003 [Lymn2003] (by means of checking a digest at execution time in the kernel to ensure that the digest matches one of the provided ones on a whitelist of programs). There are various measures that can be taken if a digest is missing from this list, or if there is a mismatch. The NetBSD Guide [NetBSD2012] outlines the ways of using veriexec, and the actions that can take place on a missing digest, or on a mismatch:

- In strict level 0, learning mode, Veriexec will act passively and simply warn about any anomalies. Combined with verbose level 1, running the system in this mode can help you fine-tune the signatures file. This is also the only strict level in which you can load new entries to the kernel.

- Strict level 1, or IDS mode, will deny access to files with a fingerprint mismatch. This mode suits mostly to users who simply want to prevent access to files which might have been maliciously modified by an attacker.
- Strict level 2, IPS mode, takes a step towards trying to protect the integrity of monitored files. In addition to preventing access to files with a fingerprint mismatch, it will also deny write access and prevent the removal of monitored files, and enforce the way monitored files are accessed. (as the signatures file specifies).
- Lockdown mode (strict level 3) can be used in highly critical situations such as custom made special-purpose machines, or as a last line of defense after an attacker compromised the system and we want to prevent traces from being removed, so we can perform post-mortem analysis. It will prevent the creation of new files, and deny access to files not monitored by Veriexec.

These are meant for different use cases. The digests are loaded with the kernel at securelevel [NetBSD2005] at a low level, and the secure level should be raised after loading the digests.

However, digests only assure that a file hashes to a certain value. It is possible that, with the advent of faster methods of performing second pre-imaging attacks, that new files can be built which will have the same digest value, but contain different contents.

If the secure level is not raised, or there is some other means of modifying the kernel's list of digests, then it is possible for the veriexec mechanism to be side-stepped.

1.2 Digital Signatures

With the advent of a BSD-licensed digital signature mechanism [Crooks2009] - which does not rely on a trusted third party such as PKI certificates to provide "trust" - we can take the digest verifying mechanism of veriexec to the next level, and use digital signatures to prove where software comes from - this is usually known as its "provenance".

A digital signature has many components, including a userid (which is generally an "email address" string and a real name denoting identity), validity dates (date valid from, date valid to), different types of document being signed, and other useful attributes. Furthermore, with key signing parties, the signatures on PGP keys provide a web of trust through which others attest to the known identity. This can be built upon to provide general attestation that the identity used to sign software is known by a diverse number of people. This identity and trust can be checked using the public key servers at pgp.mit.edu and elsewhere. New keys can be downloaded (through `gpg --recv-keys` or `hkpclient`, part of netpgp). There is no need to pay a third party to provide trust, especially when that third party may have key management issues internally, or where no further checking of identity (corporate or otherwise) takes place. In short, our trust comes from the users who have already attested to government identification documents for the individuals. Where a "role id" provides a digital signature, such as **security-officer@NetBSD.org**, the trust on that signature is backed up by the individuals who have signed the security-officer key, and the people who trust those individuals.

A signature may only be produced by someone who has both access to the private key, and who knows the passphrase guarding the encrypted key (always assuming that the key is passphrase protected). For this reason, signing of binaries and other files usually takes place on single, well-guarded and well-protected machines, not on multi-user project machines.

Because digital signatures can have validity dates, it is possible for signatures to not match because the date has not yet arrived, or the expiration date has already passed. This can be used in software licensing, but will not be covered here.

As the public key is available on public key servers, it is possible for software to ship without a valid public key, allowing the key to be installed by different means at a later date. This allows the public key to change. but has the drawback that less attention is paid when installing a new key. DNS poisoning or hijacking could be used to provide substituted identities and keys for server farms known to be installing or otherwise, including cloud instances being spun up from known-good gold master images.

Key management practices were briefly mentioned earlier, and the attack on the DSA key used by Sony to sign applications so that they could run on the PS3 [Register2010] has been the subject of much media interest. While most people have now moved on from using DSA keys (and using Elgamal to encrypt), it is worthwhile stressing that current trends have ECDSA use growing rapidly, and RSA continuing to be the signature (and encryption) used the most, from SSL certs to user keys. There are a number of reasons for this, including the need for a good entropy source at signature time for DSA, the early distrust of DSA due to rumoured trapdoor primes, its provenance from NIST, its use of 4 individual numbers making it more complex than RSA, and the use of Elgamal for encryption.

2. Identities and Trust

2.1 PGP Identities, Signing and Verification

[RFC4880] defines the PGP packet format, and the information contained therein. It defines primary identities, which are made up of one or more user identities. Each of these identities can have the identity credentials signed by other primary identities, to assure others that the identity has been matched (usually to government-issued identity documents). Often identities will cross-assure, and also endorse others credentials. From this, the PGP web of trust is built.

A primary identity is made up of a number of subkeys. The first subkey is, by convention, the key used for signing. Subsequent keys are used for encryption. Each subkey can have different user identities related to it. A user identity usually relates to an email address; it is a different personality of the key holder. A key or subkey is identified by a fingerprint - a string of hexadecimal characters the same size as a SHA-1 digest. Again, by convention, a user's PGP key is known by the first key, the one that is used for signing. User identifiers are known by (the hexadecimal representation of) a 64-bit integer, which corresponds in PGP v4 to the last 8 octets of the fingerprint of the signing key.

In turn, each key and subkey is made up of public and private parts of the key. Data is signed by using the private key to produce a BIGNUM, and teh signature is verified using the public key part of the same key.

When data is signed, a digest is taken of the document. Before the digest is finalised, some "hashed material" (principally the userid of the signer, and the signing date) is appended to the input data. The digest is finalised, and the first two octets are recorded in the signature to enable fast recognition at signature verification time. The finalised digest is then signed using the private key of the signer, and a BIGNUM is recorded as being the signature of the data. [RFC4880] differentiates between two types of input data - ordinary data, and text doocuments, which are subject to different rules for calculating digests. Gpg interprets these rules as follows: all line endings except the last one are converted from Unix-style “\n” line endings, to DOS-style “\r\n” endings. The last line ending is removed completely by gpg. This paper, and signed verification of binaries, will obviously focus on binary data.

At verification time, the digest is again computed, and the hashed material is appended before digest finalisation. The calculated digest can see quickly if it is worthwhile calculating the validity of the BIGNUM by examining the first two octets (stored in the signature). If the two octets match, the public key part is then used to match the BIGNUM representing the signature.

As such, signatures are only as secure as the underlying digest algorithm against second pre-imaging attacks (the ability to create a file with different contents which hashes to the same digest). PGP keys can contain attributes specifying many things; amongst these is the preferred digest algorithms for the key, presented in an ordered list.

2.2 Who do we trust?

To be able to use digitally signed binaries, we need to know who we can trust, and who we cannot. Usually, in the form of project-provided binaries, this is not a hard decision. However, this may not always be the case. Usually the security-officer will sign the release hashes, but will they really take the time to verify every binary which is presented to them, without knowing the provenance. Ken Thomson's 1984 ACM Turing Award lecture "Reflections On Trusting Trust" [Thomson1984] raised awareness of the role of compilers in
producing released software, and the dangers of trusting compilers blindly.

The NetBSD cross-build build.sh [Mewburn2003] script aids in this, when generating signatures for a whole operating system. In addition, it is likely that some binary packages will be needed, as well as some custom software (and immutable, important files), so a number of identities are likely to be needed when signing all necessary files on a deployed server. Armed with a fast computer, it is possible to cross-build, and cross-sign, a number of files while being assured of the provenance of these files.

The difference between the PGP web of trust (or bottom-up assurance) is different to the style of top-down assurance offered, usually as a commercial service, by Certifying Authorities, or CAs. There have been numerous examples of CAs issuing certificates - basically an entity’s public key signed by the CA) - without checking the details of the entity fully. Other concerns are the internal key management policies used by CAs which allow their own private keys to be stolen and used to issue fraudulent certificates. This whole side of things is exacerbated by the fact that Internet commerce relies on SSL and certificates, and the fact that certificate revocation policies, facilities and functionality are lacking. Again, for the purposes of controlled execution by signed binaries, we will concentrate on the web of trust model of assurance, which is a much better fit for open source software in general. It is noted, in passing, that Microsoft’s Authenticode [Microsoft2012] relies on CA-issued certificates.

2.3 Loading public keys into the kernel

The NetBSD kernel uses a veriexec mechanism to load digests into the kernel. This is done using property lists (plists) being written into the `/dev/veriexec` device from userland early in the boot process, and before raising the securelevel.

This mechanism is also used in our work to load public keys into the kernel. Because the encoding of PGP packets is done in what is usually considered an unusual way [RFC4880], and because parsing abilities in the kernel are somewhat limited, the public key information has been unrolled into its constituent parts. The netpgpkeys(1) utility was modified to have a new --trusted-keys argument, and is used to output the public key in a format which can be easily transmitted to the kernel through the `/dev/veriexec` device. The line in the public key has been wrapped below for readability:

```
% netpgpkeys --trusted-keys agc
netpgp: default key set to "c0596823"
key=d4159deb336de4cccdfa00cd1b68dcfcc0596823
name=Alistair Crooks <agc@alistaircrooks.com>
creation=1073907523
expiry=0
version=4
alg=1
n=C1D52B322E2C3BCCBDAA55A10A591DA2502B6619ED644289828C16ED75AFFB
9EE4F957B4E4D761171A139455C5F1D15320F5334ACD097E2C37844074D77A43
C2366F3BE4FFAC7454388E9B1013817D2D9735E6B45C75C9A71A8C9524C4F776
3E8849D43F8AA4E69FBCBD71A84045E92F1C6B8B3CD80AC0287D0A197DF10DC3
259C96731EDB7FE9E241A80831E6E5BEBAD3AF3AA2E5AFC55AFB4E86C475CFD0
E2BAFDF0B6213247886730B8129DDC8A05CB9DBEEC7FFE245182567D0BED3D37
B39E08474CFF1192AC7D60C04D8108A1107F1B97A5F2FAE3A48A262678F04159
30F5A2C2593DB5B0D2ADD30FA8679C5952F6601C909D1849FA0607FDC11822679D
e=29
%
```

2.4 Uses of public keys

At the present time, it is not possible to specify that certain signatures may only be used on certain files. For that reason, it is essential that people using digital signatures combined with the veriexec mechanism take great care to follow best practices for key management and signing. This is only mentioned as the loss or leakage of a private key will mean rapid re-deployment of different public keys and signatures, and this is not always possible in production environments. However, this should be contrasted with the problems suffered when certificates were found to have been malevolently issued and used [Register2011], or the problems caused by lack of useful CRLs or OSCP in popular browsers and the ensuing blacklisting of certain keys in November 2011, and the means of revocation of certain keys suffered by web portals [Register2012].

3. Digital Signatures in the Kernel

3.1 Implementation

The standard version of netpgp in the NetBSD source tree uses libcrypto for its arbitrary precision integer implementation. In May 2012, a separate branch was added, the *agc-netpgp-standalone* branch, which uses its own libraries, based on libtommath [StDenis2010]. A separate interface, to the same API as libcrypto's BIGNUM interface, was constructed, using libtommath's mpi implementation to provide all the innteger mathematics and bit-shifting.

This implementation has been added to the NetBSD kernel. In addition, the existing kernel veriexec machinery has been modified to also recognise signatures and public keys, and to parse them to retrieve the necessary information, PGP userids, validity dates and other pertinent information.

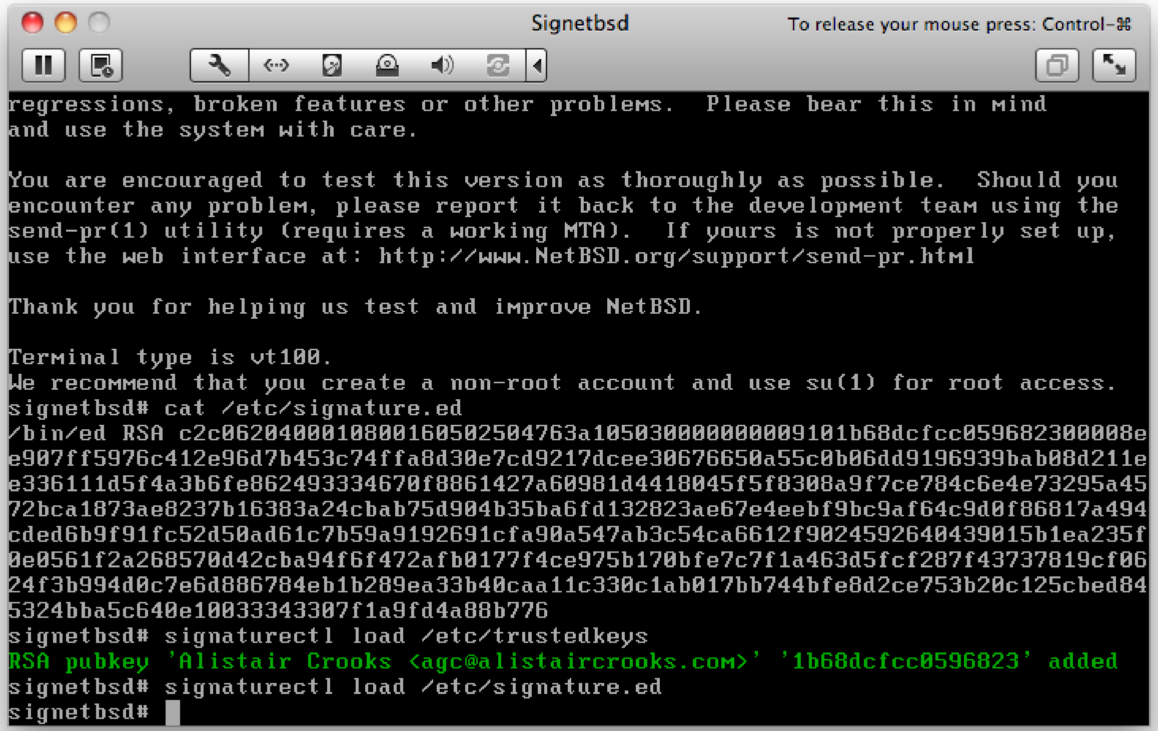
At the user level, a new command, signaturectl(1), has been constructed to load, unload, and query the kernel's implementation of digital signatures through the `/dev/veriexec` device. The functionality for this has subsequently been merged back into the veriexecctl(8) program. A command for signing multiple binaries is described later.

3.2 Signature Loading

Again, the loading of digital signatures into the kernel is accomplished using the existing veriexec and property list method of loading digests into the kernel.

Veriexec supports MD5, SHA1, SHA256, SHA384, SHA512, and RMD160 digests. To that list, we add the RSA keyword, and special processing takes in place in kern_veriexec.c when a plist object with the "RSA" keyword is received.

The following screenshot shows a public key and a signature for a single file being loaded into a kernel built with signed execution:



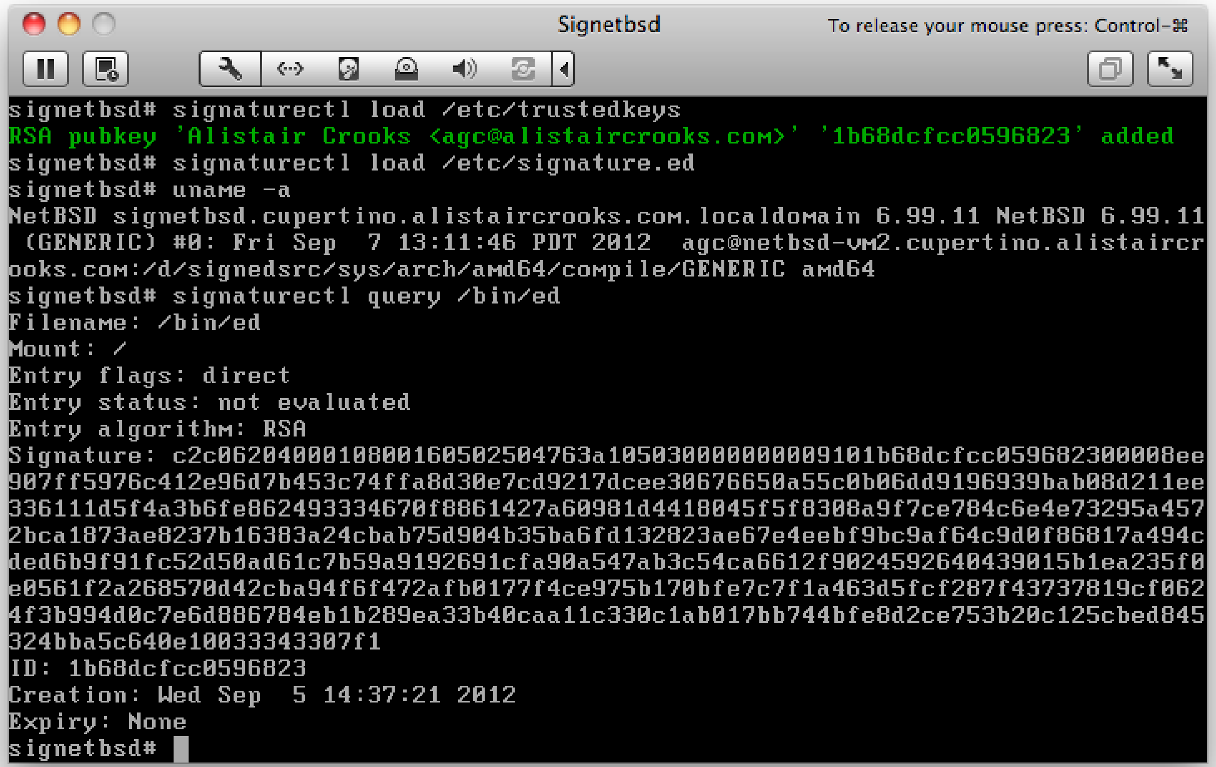
```
regressions, broken features or other problems.  Please bear this in mind
and use the system with care.

You are encouraged to test this version as thoroughly as possible.  Should you
encounter any problem, please report it back to the development team using the
send-pr(1) utility (requires a working MTA).  If yours is not properly set up,
use the web interface at: http://www.NetBSD.org/support/send-pr.html

Thank you for helping us test and improve NetBSD.

Terminal type is vt100.
We recommend that you create a non-root account and use su(1) for root access.
signetbsd# cat /etc/signature.ed
/bin/ed RSA c2c0620400010800160502504763a10503000000000009101b68dcfcc059682300008e
e907ff5976c412e96d7b453c74ffa8d30e7cd9217dcee30676650a55c0b06dd9196939bab08d211e
e336111d5f4a3b6fe8624933334670f8861427a60981d4418045f5f8308a9f7ce784c6e4e73295a45
72bca1873ae8237b16383a24cbab75d904b35ba6fd132823ae67e4eebf9bc9af64c9d0f86817a494
cded6b9f91fc52d50ad61c7b59a9192691cfa90a547ab3c54ca6612f9024592640439015b1ea235f
0e0561f2a268570d42cba94f6f472afb0177f4ce975b170bfe7c7f1a463d5fcf287f43737819cf06
24f3b994d0c7e6d886784eb1b289ea33b40caa11c330c1ab017bb744bfe8d2ce753b20c125cbed84
5324bba5c640e10033343307f1a9fd4a88b776
signetbsd# signaturectl load /etc/trustedkeys
RSA pubkey 'Alistair Crooks <agc@alistaircrooks.com>' '1b68dcfcc0596823' added
signetbsd# signaturectl load /etc/signature.ed
signetbsd#
```

Signaturectl can be used at any time to query the status of the file:



```
signetbsd# signaturectl load /etc/trustedkeys
RSA pubkey 'Alistair Crooks <agc@alistaircrooks.com>' '1b68dcfcc0596823' added
signetbsd# signaturectl load /etc/signature.ed
signetbsd# uname -a
NetBSD signetbsd.cupertino.alistaircrooks.com.localdomain 6.99.11 NetBSD 6.99.11
 (GENERIC) #0: Fri Sep  7 13:11:46 PDT 2012  agc@netbsd-vm2.cupertino.alistaircr
ooks.com:/d/signedsrc/sys/arch/amd64/compile/GENERIC amd64
signetbsd# signaturectl query /bin/ed
Filename: /bin/ed
Mount: /
Entry flags: direct
Entry status: not evaluated
Entry algorithm: RSA
Signature: c2c0620400010800160502504763a10503000000000009101b68dcfcc059682300008ee
907ff5976c412e96d7b453c74ffa8d30e7cd9217dcee30676650a55c0b06dd9196939bab08d211ee
336111d5f4a3b6fe862493334670f8861427a60981d4418045f5f8308a9f7ce784c6e4e73295a457
2bca1873ae8237b16383a24cbab75d904b35ba6fd132823ae67e4eebf9bc9af64c9d0f86817a494c
ded6b9f91fc52d50ad61c7b59a9192691cfa90a547ab3c54ca6612f9024592640439015b1ea235f0
e0561f2a268570d42cba94f6f472afb0177f4ce975b170bfe7c7f1a463d5fcf287f43737819cf062
4f3b994d0c7e6d886784eb1b289ea33b40caa11c330c1ab017bb744bfe8d2ce753b20c125cbed845
324bba5c640e10033343307f1
ID: 1b68dcfcc0596823
Creation: Wed Sep  5 14:37:21 2012
Expiry: None
signetbsd# 
```

3.3 Signing Files in Bulk

To aid in signing a large number of files, a special utility has been produced called multisign(1). It takes the credentials for the user's private key, and generates signatures for all files specified on the command line.

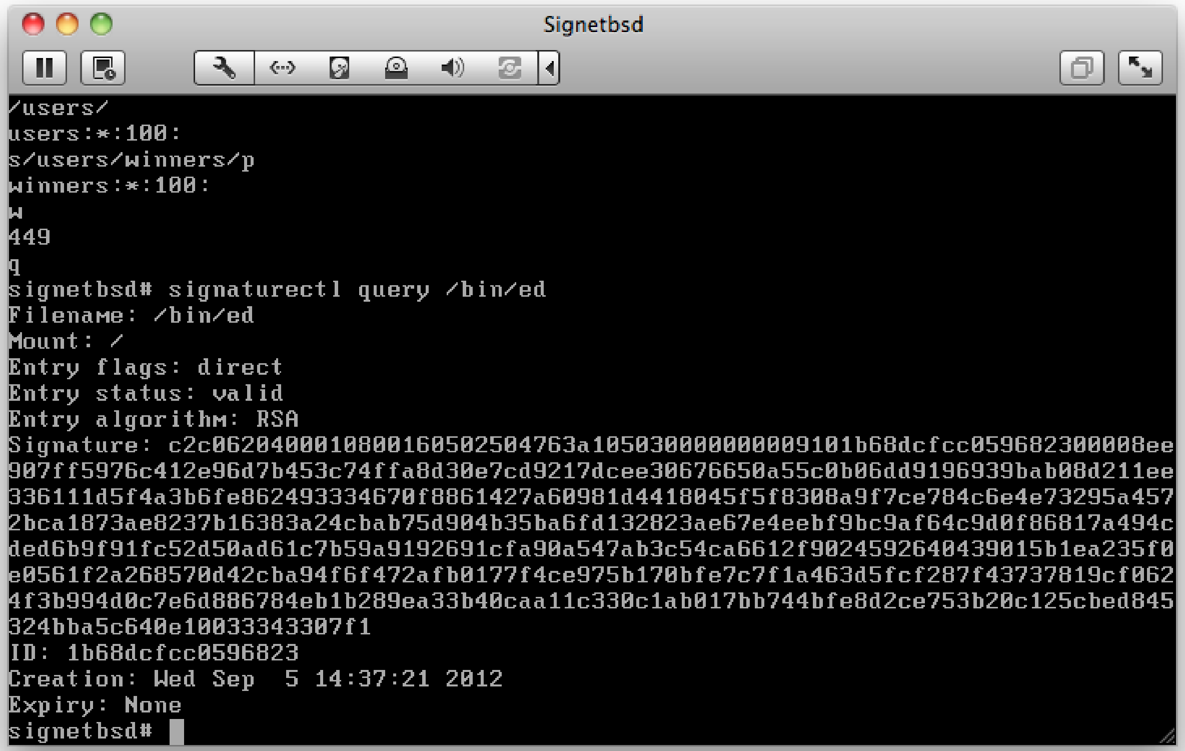
Once again, only those files whose provenance is known should be signed, since, in fact, people will be judged on the files they sign. If malware, viruses, keyloggers, trojaned files or other software is signed, trust will be lost in the signing entity. For similar reasons, keys and passphrases to those keys must be kept separate and not divulged to anyone or leaked in any way.

3.4. Signature Verification in the Kernel

At execution time, or when the file is first accessed, the veriexec mechanism calculates the digest value of the file. This is used as input to the signature calculation mechanism, and then hashed material from the signature itself is appended, and then that is used as input to the signature calculation mechanism.

If the resulting values match, then the signature has matched, and the file is considered to be the same one that was signed.

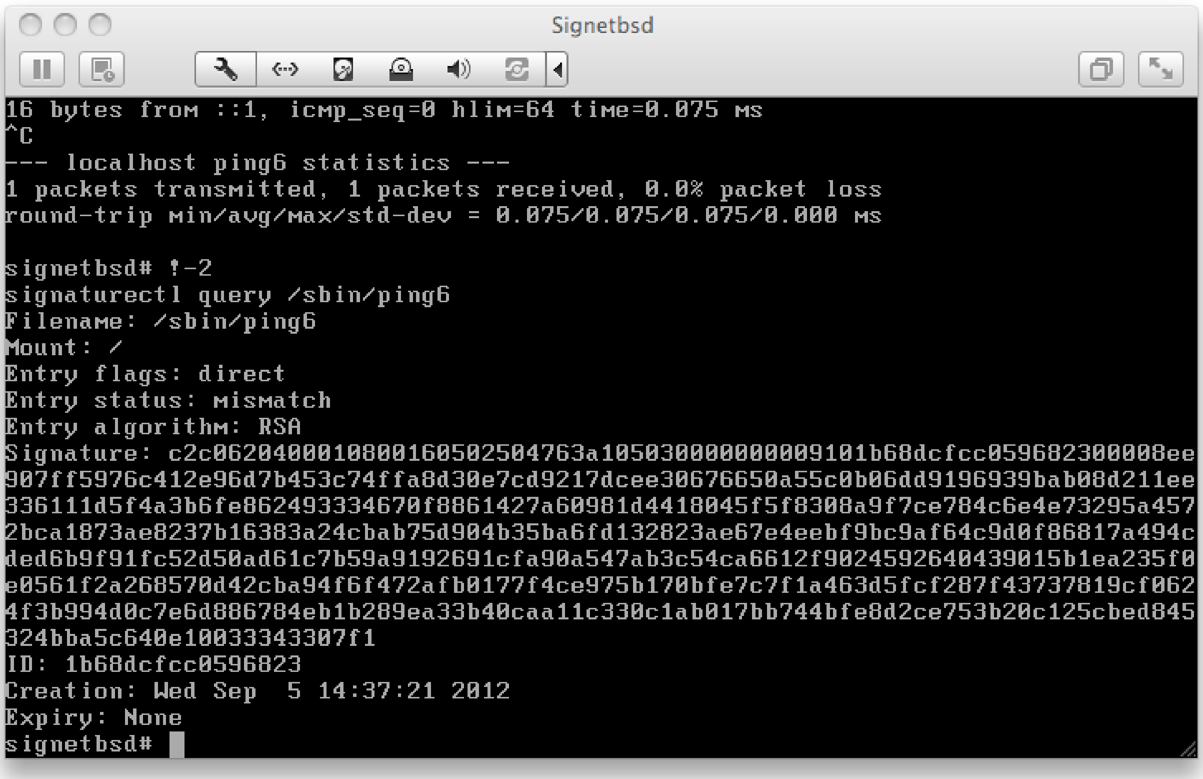
The following screenshot shows the previously loaded signature for the ed(1) binary being exercised by invoking ed.



```
/users/
users:*:100:
s/users/winners/p
winners:*:100:
w
449
q
signetbsd# signaturectl query /bin/ed
Filename: /bin/ed
Mount: /
Entry flags: direct
Entry status: valid
Entry algorithm: RSA
Signature: c2c0620400010800160502504763a105030000000009101b68dcfcc059682300008ee
907ff5976c412e96d7b453c74ffa8d30e7cd9217dcee30676650a55c0b06dd9196939bab08d211ee
336111d5f4a3b6fe862493334670f8861427a60981d44180045f5f8308a9f7ce784c6e4e73295a457
2bca1873ae8237b16383a24cbab75d904b35ba6fd132823ae67e4eebf9bc9af64c9d0f86817a494c
ded6b9f91fc52d50ad61c7b59a9192691cfa90a547ab3c54ca6612f9024592640439015b1ea235f0
e0561f2a268570d42cba94f6f472afb0177f4ce975b170bfe7c7f1a463d5fcf287f43737819cf062
4f3b994d0c7e6d886784eb1b289ea33b40caa11c330c1ab017bb744bfe8d2ce753b20c125cbed845
324bba5c640e10033343307f1
ID: 1b68dcfcc0596823
Creation: Wed Sep  5 14:37:21 2012
Expiry: None
signetbsd#
```

The signature information above is in standard terminal colour, not the green kernel printf colour, as this is information retrieved from the kernel by the signaturectl binary via proplists over `/dev/veriexec`. The validity dates of the signature are displayed, along with the "direct" status of the veriexec entry (because the ed(1) binary was directly invoked). The "valid" entry status shows that the signature was verified by the kernel signed execution mechanism.

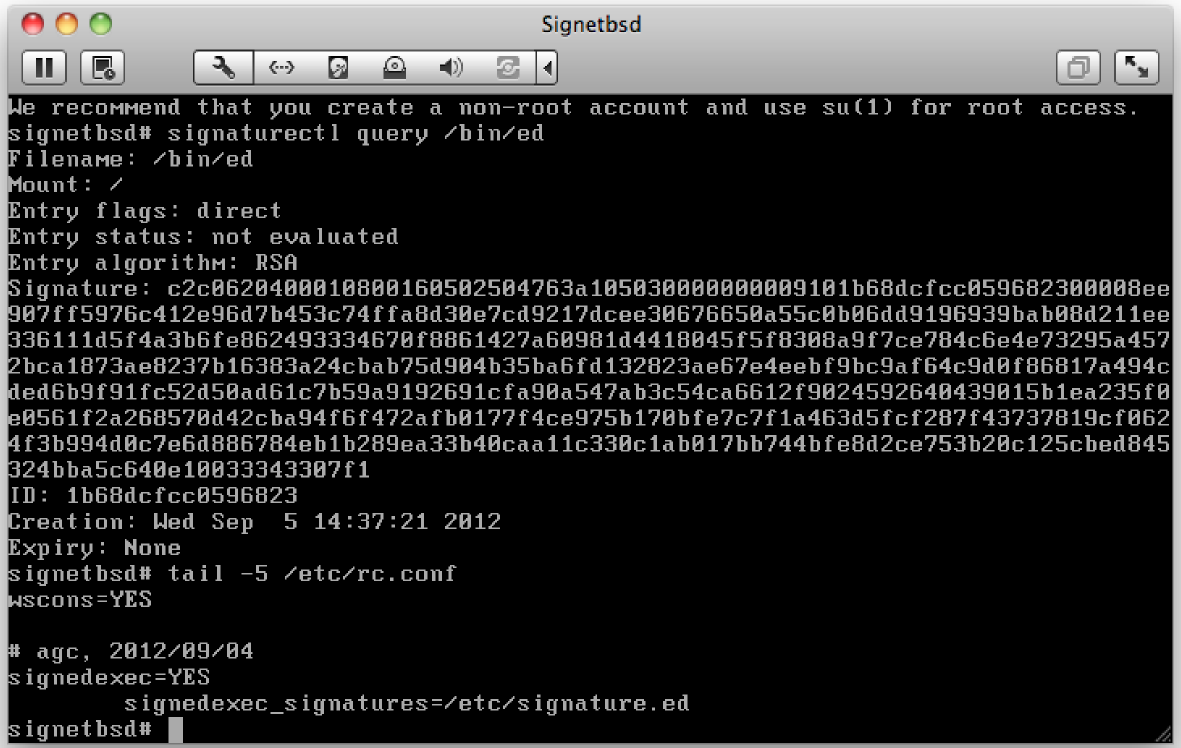
The following screenshot shows what happens when an invalid signature is read. To illustrate this, the valid signature for /bin/ed is taken and attempted to re-use for /sbin/ping6. We shall use ping6 in this illustration since experience shows that usually this binary is picked upon in systems for being relatively infrequently used, but being setuid root. In the author's experience, this binary is commonly used to carry malignant payload. We are running this in development mode (with veriexec strictness level, so no action takes place on a signature mismatch).



```
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.075 ms
^C
--- localhost ping6 statistics ---
1 packets transmitted, 1 packets received, 0.0% packet loss
round-trip min/avg/max/std-dev = 0.075/0.075/0.075/0.000 ms

signetbsd# !-2
signaturectl query /sbin/ping6
Filename: /sbin/ping6
Mount: /
Entry flags: direct
Entry status: mismatch
Entry algorithm: RSA
Signature: c2c0620400010800160502504763a10503000000000009101b68dcfcc059682300008ee
907ff5976c412e96d7b453c74ffa8d30e7cd9217dcee30676650a55c0b06dd9196939bab08d211ee
336111d5f4a3b6fe862493334670f8861427a60981d4418045f5f8308a9f7ce784c6e4e73295a457
2bca1873ae8237b16383a24cbab75d904b35ba6fd132823ae67e4eebf9bc9af64c9d0f86817a494c
ded6b9f91fc52d50ad61c7b59a9192691cfa90a547ab3c54ca6612f9024592640439015b1ea235f0
e0561f2a268570d42cba94f6f472afb0177f4ce975b170bfe7c7f1a463d5fcf287f43737819cf062
4f3b994d0c7e6d886784eb1b289ea33b40caa11c330c1ab017bb744bfe8d2ce753b20c125cbed845
324bba5c640e10033343307f1
ID: 1b68dcfcc0596823
Creation: Wed Sep  5 14:37:21 2012
Expiry: None
signetbsd# 
```

Controlled execution of signed binaries has been integrated with the /etc/rc.d system. One of the details of veriexec itself is that, once the kernel securelevel has been raised after boot, no more changes can be made to the veriexec strictness level. This screenshot shows a query on the status of the /bin/ed binary directly after executing rc.d (including loading keys and signatures).



```
We recommend that you create a non-root account and use su(1) for root access.
signetbsd# signaturectl query /bin/ed
Filename: /bin/ed
Mount: /
Entry flags: direct
Entry status: not evaluated
Entry algorithm: RSA
Signature: c2c0620400010800160502504763a10503000000000009101b68dcfcc059682300008ee
907ff5976c412e96d7b453c74ffa8d30e7cd9217dcee30676650a55c0b06dd9196939bab08d211ee
336111d5f4a3b6fe862493334670f8861427a60981d4418045f5f8308a9f7ce784c6e4e73295a457
2bca1873ae8237b16383a24cbab75d904b35ba6fd132823ae67e4eebf9bc9af64c9d0f86817a494c
ded6b9f91fc52d50ad61c7b59a9192691cfa90a547ab3c54ca6612f9024592640439015b1ea235f0
e0561f2a268570d42cba94f6f472afb0177f4ce975b170bfe7c7f1a463d5fcf287f43737819cf062
4f3b994d0c7e6d886784eb1b289ea33b40caa11c330c1ab017bb744bfe8d2ce753b20c125cbed845
324bba5c640e10033343307f1
ID: 1b68dcfcc0596823
Creation: Wed Sep  5 14:37:21 2012
Expiry: None
signetbsd# tail -5 /etc/rc.conf
wscons=YES

# agc, 2012/09/04
signedexec=YES
        signedexec_signatures=/etc/signature.ed
signetbsd# 
```

3.5 Performance

The digest over the file only gets calculated when the file is accessed the first time. After that, the original digest calculation is used.

[Lymn2003] notes that moving from a phase where the digest was calculated everytime the file was accessed, to caching the results, brought significant performance improvements:

> "The result of this caching mechanism takes a technique that made the machine run 70% slower (i.e. things took 1.7 times longer to run) to a point where the impact on the system cannot be realistically measured."

Even the additional overhead of verifying the RSA signature with a 2048 bit RSA key, which is not a heavyweight operation, makes no noticeable impact on the performance of a system.

3.6 Possible Effects of Open Source Signed Execution

It is now possible to secure more adequately edge services and cloud instances, since the bar to executing code on those machines has been raised. With a standard php installation, for example, a file's presence in the php directory means that it can be executed remotely by using the URI in a browser to cause php to open the file. In effect, the only software which will run on a machine protected by signed execution is approved software.

As mentioned earlier, the time periods of being able to run software can be specified and enforced. While this is not much use in the open source community, it is a benefit to others out there who take our work and re-use it in commercial offerings. Consider company which would like content available only after a specified time, and to have the content unavailable after an evaluation period, or a licensing term.

Some firms are known to take BSD operating systems and embed them as the operating system inside HSMs. This kind of operation is made much easier with signed execution of binaries.

It is also now possible to move some of the security away from worrying about what files are on a specific machine. IDS, in the form of crontab-based digest checkers, is not necessary on a machine where you cannot open a file unless it is a known good version. Veriexec provides us some of that, but signed execution takes that one stage further. This form of IDS can be prohibitively expensive in terms of machine cycles to run, and the lists of digests need to be kept up to date. In server farms where the machines are sized without much overhead for efficiency, it is often too complicated to run digest checking on a frequent enough basis to provide any integral security checking, and the digest database needs to be protected in order to ensure that the correct digests are used to check the files.

Different keys may be used to assure the binaries whihc are run. The NetBSD security-officer could sign the NetBSD binaries from the 6.0 release, for example. A company's packaging team could sign the packages they produce. Please note that, at the present times, pkgsrc packages and RPMs are both signed on the contents of the binary package, and so new signatures will need to be made on (each file of) the contents of any external packages which are used. It is usually the case, though, that any software release
will include binary packages built in-house specifically for the software release itself - see for example FreeBSD's use of the `nanobsd` script to produce embedded software releases.

3.7 Upgrading and Patching

For the purposes of this document, patching is the practice of adding vendor-supplied binary patches to a machine to update software on there. Upgrading is seen more as a full operating system upgrade.

The best way to update some software on a machine is by making sure that horizontal scaling is in place, and then take each instance out of rotation, and upgrade the machine.

This extends the best practice for upgrading VM instances in the cloud, where patching is not done; rather new instances are spun up from a new gold master, and the old instance is retired.

3.8 Code Availability

The code for this is now, or will be available in the very near future, on a branch in the NetBSD code repository.

4. Future Work

It is hoped that the existing mechanism for verifying digital signatures within the kernel can be extended to other parts of the kernel. For instance, signed system calls (the verification of data provided as the argument to a system call) can easily be implemented, with the verification of the identity of the user now moving from user level binaries and libraries to inside the kernel.

5. Related Work

As has been described above, the controlled execution of signed binaries is an extension of Lymn's veriexec subsystem. It is a common element in mobile phones, games machines and other personal devices - see below.

Various commercial entites have closed source implementations of signed binary execution.

Ubuntu Linux (10.04 LTS, Lucid Lynx) has an elfsign/elfverify feature built around CA certs [Elfsign2012] and [ElfSignProcedure2012]

Microsoft have a CA cert-based implementation of signed code execution called Authenticode [Microsoft2012]

Apple's Mountain Lion release includes functionality called Developer Id and Gatekeeper [Apple2012]. iPhone apps have had this for a longer time.

Sony's PS3 uses code signing to allow applications to run on PS3s, and it was reported to have been hacked (by brute force attacks) late in 2010 [Register2010]

6. Conclusion

We have shown that, building on top of the existing veriexec framework, an additional means of using digital signatures is used to verify the open(2) of any file on a machine. If the contents of the file do not match the expected contents, as signed by a person in a position of trust in generating that software, the file will not be opened or used.

7. References

Apple2012 https://developer.apple.com/resources/developer-id/

Crooks2009 http://www.ukuug.org/events/eurobsdcon2009/papers/netpgp.pdf

Elfsign2012 http://www.hick.org/code/skape/papers/elfsign.txt

ElfSignProcedure2012 http://us.generation-nt.com/answer/want-sign-verify-binary-using-elfsign-pls-let-me-know-procedure-help-205955331.html

Lymn2003 http://www.users.on.net/~blymn/veriexec/

Mewburn2003 http://www.mewburn.net/luke/papers/build.sh.pdf

Microsoft2012 http://technet.microsoft.com/en-us/library/cc750035.aspx

NetBSD2012 http://www.netbsd.org/docs/guide/en/chap-veriexec.html

Register2010 http://www.theregister.co.uk/2010/12/30/ps3_jailbreak_hack/

Register2011 http://www.theregister.co.uk/2011/08/29/fraudulent_google_ssl_certificate/

Register2012 http://www.theregister.co.uk/2012/05/24/yahoo_ships_private_certificate_by_accident/

RFC4880 http://tools.ietf.org/html/rfc4880

StDenis2010 http://libtom.org/

Thomson1984 http://cm.bell-labs.com/who/ken/trust.html