OpenSSL API

This demo is based on CVE-2008-5077:

> *OpenSSL 0.9.8i and earlier does not properly check the return value from the EVP_VerifyFinal function, which allows remote attackers to bypass validation of the certificate chain via a malformed SSL/TLS signature for DSA and ECDSA keys.*

Core OpenSSL apps such as `openssl verify` — not just third-party libraries — misused the OpenSSL libraries, treating a tri-state return code (success, error, fail) as a simple boolean value. This led to exceptional failures being treated as successes in fourteen places in the OpenSSL codebase, for example:

```
diff -ur openssl-0.9.8i-ORIG/apps/verify.c openssl-0.9.8i/apps/verify.c
--- openssl-0.9.8i-ORIG/apps/verify.c   2004-11-29 11:28:07.000000000 +0000
+++ openssl-0.9.8i/apps/verify.c        2008-12-04 00:00:00.600000000 +0000
@@ -266,7 +266,7 @@

        ret=0;
 end:
-       if (i)
+       if (i > 0)
                {
                fprintf(stdout,"OK\n");
                ret=1;
```

The demo located in $TESLA_SOURCE_DIR/demo/openssl-api replicates this verification error and demonstrates how TESLA can detect it. `openssl-api.c:176` contains the following erroroneous code:

```
int result = X509_verify_cert(ctx);
if (result)    // should test (result == 1)
        success = true;
else
        *error = X509_verify_cert_error_string(ctx->error);
```

`openssl-api.c:201` shows how a consumer of X.509 certificates can use TESLA to defensively check for such an incorrect verification with an inline temporal assertion:

```
void
use_cert(X509 *cert)
{
#ifdef TESLA
        /*
         * Assert that the certificate has been properly verified.
         */
        TESLA_WITHIN(main, previously(X509_verify_cert(ANY(ptr)) == 1));
#endif
```

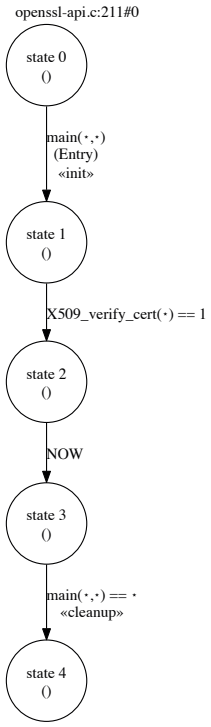
Running the demo with default settings erroneously verifies signatures that do not exist. In the demo setup, there are three certificate files: self-signed.crt, which is a self-signed certificate, other.crt, which is not involved in any signatures and hacked.crt, which is not a valid certificate at all: it has a valid form but its key fingerprint is incorrect.

In the demo, OpenSSL's X509_verify_cert() is being fed invalid data, so when running without TESLA, the demo "verify the signature" routine always returns true:

```
[TESLA/demos/openssl-api]$ ./demo
[...]
'Inputs/self-signed.crt' is signed by 'Inputs/self-signed.crt'.
'Inputs/self-signed.crt' is signed by 'Inputs/other.crt'.
'Inputs/self-signed.crt' is signed by 'Inputs/hacked.crt'.
```

With TESLA, however, the error is detected: TESLA expects to match the `X509_verify_cert(ANY(ptr)) == 1` event, which does not occur in this case. When `X509_verify_cert` returns -1, the event does not match:

```
[TESLA/demos/openssl-api]$ ./demo -D TESLA
[...]
TESLA failure:
In automaton 'openssl-api.c:211#0':
automaton 0 {
        state 0: --(main(X,X): Entry)-->(1 <<init>>)
        state 1: --(X509_verify_cert(X) == 1)-->(2)
        state 2: --(NOW)-->(3)
        state 3: --(main(X,X) == X)-->(4 <<cleanup>>)
        state 4:
}
openssl-api: Instance 0 is in state 1
but required to take a transition in [ (2:0x0 -> 3) ]
```



The error output (above) indicates that TESLA expected to take a transition from state 2 to state 3 because of a NOW event: the program execution has reached the site of the inline temporal assertion. This transition (also illustrated to the right), cannot be taken because the automaton instance is still in state 1: TESLA has not matched the function return event `X509_verify_cert(ANY(ptr)) == 1`.

Audit

This demo shows how TESLA can detect incorrect — and very indirect — usage of a security-relevant API, in this case the FreeBSD auditing framework.

Auditing, like many security functions, is a cross-cutting concern: its implementation cuts across the abstraction boundaries of typical procedural or object-oriented programs. Implementation errors in one module may violate assumptions in another module.

For instance, the following code (from FreeBSD's `sys/kern/vfs_vnops.c:812`) might assume that, before writing to the filesystem, all appropriate filename audit records have been created:

```
static int
vn_write(fp, uio, active_cred, flags, td)
        struct file *fp;
        struct uio *uio;
        struct ucred *active_cred;
        int flags;
        struct thread *td;
{
  /* ... */
  error = VOP_WRITE(vp, uio, ioflag, fp->f_cred);
  /* ... */
}
```

The validity of this assumption depends on the implementation of the file name → filesystem vnode conversion performed by the `namei()` function at `sys/kern/vfs_lookup.c:213`:

```
if (cnp->cn_flags & AUDITVNODE1)
  AUDIT_ARG_UPATH1(td, ndp->ni_dirfd, cnp->cn_pnbuf);
if (cnp->cn_flags & AUDITVNODE2)
  AUDIT_ARG_UPATH2(td, ndp->ni_dirfd, cnp->cn_pnbuf);
```

which is itself governed by flags set elsewhere, as in this simplified model of the `abort()` syscall:

```
/* arguments to pass to namei: */
struct nameidata nd;
nd.ni_dirp = "/path/to/coredump";

struct componentname *c = &nd.ni_cnd;
c->cn_thread = curthread;
c->cn_flags = 0;     /* ERROR: should be AUDITVNODE1 */

error = namei(&nd);
if (error != 0)
  return (error);

/* write the core dump to the result vnode: nd.ni_vp */
```

TESLA allows the programmer of the filesystem interface to assert that, previously in the current system call, a path name was audited. This assertion can be written without any knowledge of the path itself, as shown in `$TESLA_SOURCE_DIR/demos/audit/vfs_vnops.c:143`:

```
static int
vn_write(fp, uio, active_cred, flags, td)
        struct file *fp;
        struct uio *uio;
        struct ucred *active_cred;
        int flags;
        struct thread *td;
{
#ifdef AUDIT
#ifdef TESLA
  TESLA_WITHIN(syscall, previously(
    called(audit_arg_upath1, td, ANY(int), ANY(ptr))));
#endif
#endif
  /* ... */
  error = VOP_WRITE(vp, uio, ioflag, fp->f_cred);
  /* ... */
}
```

When the demo is run normally (without TESLA), the audit event is silently missed:

```
[TESLA/demos/audit]$ ./demo
[...]
-- looking up name ‘/path/to/coredump’
-- writing to vnode with tag ‘found_by_namei’
```

With TESLA, however, the audit event's absence is conspicious: TESLA attempts to look up an automaton instance named by the tuple (`td=42`,φ, φ, φ) and move it from state 2 to state 3, but there is no such instance because the `audit_arg_upath1()` event has not occurred.

```
[TESLA/demos/audit]$ ./demo -D TESLA
[...]
-- looking up name ‘/path/to/something’

TESLA failure:
In automaton ‘vfs_vnops.c:146#0’:
automaton 0 {
        state 0: --(syscall(): Entry)-->(1 <<init>>)
        state 1: --(audit_arg_upath1(td,X,X): Entry)-->(2)
        state 2: --(NOW)-->(3)
        state 3: --(syscall() == X)-->(4 <<cleanup>>)
        state 4:
}
audit-demo: No instance matched key ‘0x1 [ 42 X X X ]’ for transition(s) [ (2:0x1 -> 3) ]
```