Four Six Lightning Talks (and a long one)

Ben Laurie
ben@thebunker.net
The Talks

• Why Open Source Vendors are Bad for Security
• “Be Liberal in What you Accept” - NOT!
• DNSSEC
• PEIM
• OpenPGP/SDK
• Anonymous Presence
• CaPerl
Why Open Source Vendors are Bad for Security
What do OS Vendors do?

- Package and bundle existing software
- Modify it to comply to their own vision
- Backport security fixes to old versions
  - and fail to indicate that they have done so
What do OS Vendors do?

- Break security fixes during packaging
- Introduce security flaws during packaging
Disclosure Classification

- Zero-days
- Disclosure via the author of the software
- Disclosure via a packager of the software
- Disclosure via some co-ordinator
Flavours of Co-ordinators

- CERT
- NISCC
- linux-vendors
The CERT/CC is a major reporting center for Internet security problems. Staff members provide technical advice and coordinate responses to security compromises, identify trends in intruder activity, work with other security experts to identify solutions to security problems, and disseminate information to the broad community. The CERT/CC also analyzes product vulnerabilities, publishes technical documents, and presents training courses.
NISCC

NISCC has no regulatory, legislative or law enforcement role; it seeks to achieve its aim through four broad work streams:

- **Threat Assessment.** Using a wide range of resources to investigate, assess and disrupt threats.
- **Outreach.** Promoting protection and assurance by encouraging information sharing, offering advice and fostering best practice.
- **Response.** Warning of new threats; advising on mitigation; managing disclosure of vulnerabilities; helping the CNI investigate and recover from attack.
- **Research and Development.** Devising the most advanced techniques and methods to support efforts across all work streams.
linux-vendors

- Secret mailing list
- Not just Linux - at least FreeBSD too
The Role of Co-ordinators
(thory)

• Initial point of contact
• Mediate between authors, packagers, disclosers and “critical” users
• Co-ordinate fix releases
• Manage public notification
The Role of Co-ordinators (practice)

- Protect their stakeholders
- Extend timelines to enable commercial vendors to complete lengthy internal processes
- Completely fail to help the clueful (i.e. those who can roll their own builds)
Zero-days

- The world gets the announcement simultaneously
- The authors fix the software and release a patch
- Packagers then incorporate that patch and release their own updates
- Users of the base package get fixed first
Disclosure via Author

- Author is notified, produces fix in conjunction with the attacker
- Two variants
  - Author notifies co-ordinator and/or packagers
  - Author releases patch to the world
- Users of the base package and maybe some packagers get fixed first
Disclosure via Vendor

- Vendor is notified, produces fix in conjunction with author (maybe) and notifier
- May notify some other vendors and/or co-ordinator
- Users of the base package and some packagers get fixed first
Disclosure via Co-ordinator

- Co-ordinator is notified, works with author to produce advisory and fixes

- If the co-ordinator has their way, advisory and fix is circulated to the favoured few first

- The co-ordinator does not always win this fight

- Favoured few get fixed first, followed by the rest of the world (base package users first, of course)
Why are Vendors a Problem?

• Rather than make it easy for users to take a new version of software directly from the authors, they patch it and package it - this introduces delay

• Even worse, attempting to use the base package is usually untenable, or even destructive
Why are Vendors a Problem?

- Vendors create the need for coordination - if everyone could use the base package, everyone would get fixed at the same time.
- Vendors create the myth that they are needed for reliability.
What’s the Fix?

• Packagers should work with authors to make their packages redundant - or at least minimal

• Packagers should figure out how to help their users, rather than how to make themselves indispensable
Be Liberal In What You Accept - NOT!
“In general, an implementation should be conservative in its sending behavior, and liberal in its receiving behavior.”
RFC 760, Internet Protocol, Ed. Jon Postel
A Cisco Secure Access Control Server (ACS) that is configured to use Extensible Authentication Protocol-Transport Layer Security (EAP-TLS) to authenticate users to the network will allow access to any user that uses a cryptographically correct certificate as long as the user name is valid. Cryptographically correct means that the certificate is in the appropriate format and contains valid fields. The certificate can be expired, or come from an untrusted Certificate Authority (CA) and still be cryptographically correct.
HTTP Request Smuggling

POST http://SITE/foobar.html HTTP/1.1
Host: SITE
Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 0

Proxy skips this

GET /poison.html HTTP/1.1
Host: SITE

Content-Length: 44

Webserver sees this

GET http://SITE/page_to_poison.html HTTP/1.1
Host: SITE
Bla: GET http://SITE/page_to_poison.html HTTP/1.1
Host: SITE
Connection: Keep-Alive

And sees this

And skips this
HTTP Request Smuggling

- More fun if the webserver is using name-based virtual hosts
- Even more fun if the webserver has proxying enabled (Apache under some Linux distros does by default!)
TCPA

- The plan is to have TCPA in all sorts of components
  - Ethernet cards
  - VGA cards
  - Disk drives
- I leave you to ponder the implications of multiple implementations of TCPA
DNSSEC

- DNSSEC is effectively signed DNS
- Signatures are hierarchical, starting at the root
- All responses are signed, including negative responses
- No online signing allowed!
What Does it Look Like?

www.somewhere.org IN A?

org. IN KEY ...
org. IN SIG ...
somewhere.org. IN KEY ...
somewhere.org. IN SIG ...
www.somewhere.org. IN A 1.2.3.4
www.somewhere.org. IN SIG ...
Problems

- Response size
- "Islands of Trust"
- Key rollover
- Negative responses lead to zone walking
Response Size

- Responses are BIG
- Big responses need TCP
- Many firewalls block TCP port 53
Islands of Trust

- Until everything is signed, there will be zones that have no parent signature
- Three solutions
  - Every resolver has every island’s root keys
  - Cross-certification
  - Domain Lookaside Validation
Key Rollover

- How do we expire old keys and introduce new ones?
- Only a problem at the root(s)
- Two solutions
  - Resolvers are manually reconfigured
  - Threshold signing
- Amazingly, this is still an open problem
Negative Responses

nonexistent.somewhere.org IN A?
/org. IN KEY ...
/org. IN SIG ...
somewhere.org. IN KEY ...
somewhere.org. IN SIG ...
mail.somewhere.org. NSEC A www.somewhere.org.
mail.somewhere.org. SIG ...
Walking the Zone

mail.somewhere.org IN A 1.2.3.4
ns1.somewhere.org IN A 2.3.4.5
ns2.somewhere.org IN A 3.4.5.6
www.somewhere.org IN A 4.5.6.7
Walking the Zone

a.somewhere.org. IN A?
somewhere.org. IN NSEC A mail.somewhere.org.

mail0.somewhere.org. IN A?
mail.somewhere.org IN NSEC A ns1.somewhere.org.

ns10.somewhere.org. IN A?
ns1.somewhere.org. IN NSEC A ns2.somewhere.org.

ns20.somewhere.org. IN A?

www0.somewhere.org. IN A?
The Fix

nonexistent.somewhere.org. IN A?

H,www.somewhere.org)=AF72B....
H(nonexistent.somewhere.org)=C01593...
H(ns.somewhere.org)=F2178....

AF72B.... IN NSEC3 A F2178....
AF72B.... IN SIG ...
Privacy Enhanced Identity Management

- Google Summer of Code
- Caroline Sheedy - Dublin City University
- Mentored by the Apache Software Foundation
- A library of building blocks for zero knowledge proofs
Zero Knowledge 101

• Prover proves a fact in (near) zero knowledge
• Verifier verifies the fact
• Certifier may certify the fact
• Auditor may audit the proof
Where’s Waldo?

- I want to prove I know where Waldo is, without revealing the location.
Zero Knowledge 101
Bit Commitment

- Not exactly ZK, but related
- No Certifier
- Prover chooses a bit, and commits to it
- Later the Prover reveals the bit, and the Verifier can verify no cheating
What We’re Implementing

- Several bit commitment schemes
- ZK proof that we hold a signed certificate without revealing the certificate
- ZK proof that we know a member of a set without revealing which member
- ZK proof that we have a signed number $X$ s.t. $X > Y$, without revealing $X$. 
What the Library Provides

- Low-level functions to do the tricky crypto goop
- No protocols
- No marshalling
OpenPGP/SDK

• Free C implementation of OpenPGP
• Sponsored by Nominet
• BSD licensed
• Co-author: Rachel Willmer
Goals

- Complete
- Flexible
- Storage agnostic
- Protocol agnostic
- Correct
Functionality

- Parse OpenPGP packets into in-memory data structures
- Decompress compressed packets and parse the contents
- Call the application back for each complete packet
- Check signatures
Functionality

- Construct OpenPGP packets
- Compress them
- Sign them
Example: Read Secret Key

```c
static ops_secret_key_t skey;

static ops_parse_callback_return_t
callback(const ops_parser_content_t *content,void *arg_)
{
    if(content->tag == OPS_PTAG_CT_SECRET_KEY)
    {
        memcpy(&skey,&content->content.secret_key,sizeof(skey);
        skey_found=ops_true;
        return OPS_KEEP_MEMORY;
    }

    return OPS_RELEASE_MEMORY;
}

return OPS_RELEASE_MEMORY;
```
Example: Read Secret Key

```c
void get_key(const char *keyfile)
{
    ops_reader_fd_arg_t arg;
    ops_parse_options_t opt;

    ops_parse_options_init(&opt);
    opt.cb=callback;

    arg.fd=open(keyfile,O_RDONLY);
    assert(arg.fd >= 0);
    opt.reader_arg=&arg;
    opt.reader=ops_reader_fd;

    ops_parse(&opt);

    ....
}
```
Example: Write Public Key

ops_write_struct_public_key(&skey.public_key,&opt);
ops_fast_create_user_id(&id,user_id);
ops_write_struct_user_id(&id,&opt);
ops_signature_start(&sig,&skey.public_key,&id,
OPS_CERT_POSITIVE);
ops_signature_add_creation_time(&sig,time(NULL));
ops_keyid(keyid,&skey.public_key);
ops_signature_add_issuer_key_id(&sig,keyid);
ops_signature_add_primary_user_id(&sig,ops_true);
ops_signature_hashed_subpackets_end(&sig);
ops_write_signature(&sig,&skey.public_key,&skey,&opt);
ops_secret_key_free(&skey);
Anonymous Presence
Objectives

• Alice wants to rendezvous with Bob

• Alice doesn’t want anyone to know she is talking to Bob

• Alice and Bob don’t want their conversations to be linked

• A rendezvous server is required

• There is a global passive adversary

• The server is not trusted
Apres

- A system to achieve this
- Alice and each of her friends share a secret
- At each time slice, Alice calculates $H(\text{secret}, \text{timeslice})$ for each friend
- Alice sends the list to the server
Apres

- The server finds matching values and relays messages between them
- Alice and Bob set up a secure session based on the shared secret (e.g. modified Diffie-Hellman)
- They can then talk via the server, or directly if they choose
Unlinkability

• Since only Alice and Bob know the secret, $H(\text{secret}, \text{timeslice})$ gives unlinkability

• However, when the time changes, they mustn’t switch to the new value synchronously

• Solution: always start new timeslices a random time before they are due, and stop a random time after they expire
Unlinkability

- Alice is identifiable by number of friends
- Solution: add dummies up to the next round number
Implementation

- Apres exists as a Perl library
- Test versions for plain TCP and IRC
- IRC uses a bot and a plugin
CaPerl: Running Hostile Code Safely

or

Capabilities in Perl
Ben Laurie
The Bunker

ben@thebunker.net
Summary

Motivation

What are capabilities?

Why are they good?

History lesson

Capability Languages

CaPerl
Motivation

Collaboration means sharing - code, too

Sharing code is dangerous

I want to be able to run scripts I know nothing about
The Traditional Approach

Sandboxes, jails ... call them what you will

But sandboxes are either too restrictive or too lax

One per script? Are you kidding?

There must be a better way!

I claim there is: capabilities
What’s a Capability?

- Not a POSIX capability
- An opaque thingy that represents the ability to do something

For example...
- Read /some/file/somewhere
- Write /some/other/file/somewhere/else
- Talk on the socket connected to somehost:someport
- Put money into Ben’s bank account
- Take money out of Ben’s bank account
Why are capabilities good?

- Environment can choose exactly what the visiting code can do
- New capabilities can be invented by writing (easy) code
- Designation is authorisation
- Security partitions within code are easy
Why not use the traditional approach?

- It's hard to express "allow the program to write the file the user chose, and no other"

- You do everything twice
  - It's OK to do X
  - Please do X
Why not use the traditional approach?

You do everything thrice - BACKWARDS!

- Please do X
- Damn! Permission denied
- Its OK to do X
- Please do X
Why not use the traditional approach?

Or ... you don’t do it at all.
If capabilities are so great, why aren’t we all using them?
History

- Invented in the 60s
- Implemented in hardware, 1970-77
- Hardware was slow and expensive
- Shot down by academics - “just ACLs in disguise”
History (Operating Systems)

- KeyKOS
- EROS
- Coyotos
- CapROS
- Amoeba
The Granovetter Diagram

Alice → foo → Bob
Alice → Carol
Bob → Carol
Capability Languages

- Requirements for capabilities
- Implementing capabilities in standard languages
- Designing the language around capabilities
- Modifying standard languages
Language requirements

Capabilities are represented by opaque thingies
- An object
- A bound function
- A random number
- A pointer

Capabilities cannot be forged
Capabilities cannot be discovered
Authority is confined to capabilities
Only trusted code can create capabilities with new powers
Summary: if you don’t have the capability, you can’t do it
Implementing capabilities in standard languages

- Opacity is a problem
- Forgeability is a problem
- Confining authority is a problem
- But ... we already know which languages people like
How about C?

- Out of the question!
- All code has free access to all RAM
- No way to confine authority
- No way to prevent forging
C++?

For our purposes, the same as C
Python?

Perhaps proxies work

It may be true, but they’re horribly awkward to use

Restricted execution mode is not maintained
Perl

- Forget it!
- Way too clever
- Can load arbitrary binary
- Possibly clever overloading would work, but...
  ...would we ever trust it?
Smalltalk?

Squeak-E, apparently
Ruby? Erlang? Haskell?

I have no idea!

Not widely used
Build the language around capabilities

Several advantages

- Safety
- Ease of use
- Language fits the problem

But how many new languages succeed?
Very cool language
Deferred evaluation
Cunning network tricks
def calcMilesBeforeEmptyVow(carRcvr) {
    def [milesBeforeEmptyPromise, milesBeforeEmptyResolver] := Ref.promise()
    def fuelRemainingVow := carRcvr <- getFuelRemaining()
    def mpgVow := carRcvr <- getEfficiency()
    when (fuelRemainingVow, mpgVow) -> done(fuelRemaining, mpg) {
        milesBeforeEmptyResolver.resolve(mpg * fuelRemaining)
    } catch prob {milesBeforeEmptyResolver.smash("Car Lost" + prob)}
    return milesBeforeEmptyPromise
}
What’s wrong with E?

- Written by academic language nerds
- Esoteric syntax
- Monthly syntax changes
- Based on Java
Modifying existing languages

- Risk reduction
- Shallow learning curve
- My preferred option
- Not always possible
Modify the interpreter

Interpreter has no clue what is going on

Python developers not interested

Restricted execution mode is broken...

...and no-one cares.
Inspiration! Use the language itself to enforce capability discipline
CaPerl: outline

What is CaPerl?

Language modifications

“Taming”

Using CaPerl

Implementation

Examples
What is CaPerl?

- Converts Perl into a capability language
- Compiles into standard Perl
- Compiler is written in Perl and yapp
Language modifications

We introduce the notion of trusted vs. untrusted code
Language modifications

- public sub ...
  - anyone can call it

- trusted sub ...
  - only trusted code or its own module can call it

- sub ...
  - only its own module can call it
Language modifications

---
use module;

--- module must have been compiled by CaPerl

---
use untrusted module;

--- Only trusted code can do this

--- module can be anything
### Language modifications

- bless $self,$class;
  - Only trusted code can do this
- bless $self;
  - Untrusted code can do this
**Trusted Code**

- Can do almost all standard Perl stuff
- Can only look inside objects belonging to the same module
- Can import any module
- Can call any function in the same module
- Can call both **public** and **trusted** functions in other modules
Untrusted Code

- Can not call plain (i.e. non O-O) functions
- Can only look inside objects belonging to the same module
- Can only import trusted modules compiled with CaPerl
- Can call any function in the same module
- Can only call public functions in other modules
Taming

- Wrap libraries with capabilities
- Only trusted code can tame (obviously)
- Then we can use (almost) all of CPAN!
- **WARNING:** you have to use your brains to tame
Implementation

- Yapp and Perl create a parse tree
- Standard Perl emitted directly from the tree
- Simple recursion, small amount of context passed down the tree
- Surprisingly small
- Almost all trivial
Using CaPerl

Compile code: `caperl module.tm`

Tell the compiler whether each module is trusted or untrusted

The output is Perl, which you run in the normal way, with the CaPerl libraries on the LIBPATH

Modules are completely independent of each other
Innards

- Marking CaPerl code
- Restricting function calls
- Making objects opaque
- Loading only appropriate (i.e. trusted/untrusted) modules
- Control of function calls
Marking CaPerl code

package module;

becomes

package module;
$module::CAPERL_CODE=1;
$module::CAPERL_TRUSTED=1;
Restricting function calls

$obj->func();

becomes

(my($T1,$T2), $T1=$obj, $T2=ref($T1), (eval "\$\$\{$T2}::CAPERL_CODE" || croak 'not a capability'), $T1)->func();
Restricting function calls

sub func {
    my ($Tp,$Tf,$Tl)=caller;
    confess "Attempt to call private function from $Tf($Tl)" if $Tp ne 'module';
}
trusted sub func {

becomes

sub func {
    my ($Tp,$Tf,$Tl)=caller;
    confess "Attempt to call trusted function from $Tf($Tl)" if $Tp ne 'module' && !eval \"\${Tp}::CAPERL_TRUSTED\";
Restricting function calls

public sub func {

becomes

sub func {


Making objects opaque

$obj->{thing}

becomes

(my ($Tleft,$Tr),($Tleft=$obj,
 $Tr=ref($Tleft),$Tr eq 'ThisModule'
 || $Tr eq 'HASH'
 || die "attempt to dereference object of type
 "$Tr""),
 $Tleft)->{thing}
Loading modules

use somemodule;

becomes

use somemodule;
$somemodule::CAPERL_CODE
|| croak 'somemodule is not CaPerl code';
use untrusted module;

becomes

use module;
Examples

- Run untrusted CGIs
- Tame opendir
use untrusted CGI;
use Wrap::CGI;
use untrusted CGI::Carp;
use Wrap::POSIX;

my $cgi=new Wrap::CGI(new CGI());

$cgi->print($cgi->header(-type => 'text/html'));
my $e=\%ENV;
my $path=$e->{PATH_INFO};
my $base=$e->{WRAP_CGI_BASE};
croak('Bad path info: '.$path) if $path !~ /^\[/([A-Za-z0-9_]+)+$/;
$path=$1;
Run untrusted CGIs

my $dir="../wrapped-dirs/$path";
mkdir($dir,0700) || $! == $EEXIST ||
croak("Can't create subdir $dir: $!");

use CAP::RWDirectory;
my $dc=new CAP::RWDirectory($dir);

use untrusted File::Slurp;
my $text=read_file("$base/$path.pm");
eval($text);

my $o=eval("new $path");
$o->go($cgi,$dc);
package Wrap::opendir;

use untrusted Symbol;

trusted sub new {
  my $class=shift;
  my $dir=shift;

  my $self={};
  bless $self,$class;
  $self->{dir}=$dir;

  return $self;
}


sub DESTROY {
    my $self=shift;

    $self->{handle}=`close`;
    if defined($self->{handle});
}

public sub opendir {
    my $self=shift;

    $self->{handle}=gensym();
    return opendir($self->{handle},$self->{dir});
}
public sub readdir {
    my $self=shift;

    croak() if !$self->{handle};
    return readdir($self->{handle});
}

public sub closedir {
    my $self=shift;

    my $ret=closedir($self->{handle});
    delete($self->{handle});
    return $ret;
}
Where have I punked?

Actually, not in very many places

- Language support is incomplete
- Maybe should be “native” modules that can do plain Perl
- Loading of modules should be through capabilities
Conclusions

- Capabilities are not as hard to implement as you might think
- Capabilities are not as hard to use as you might think
- Using pre-compilation to add capabilities to existing languages is not only viable, it's relatively easy
- Capabilities have powerful security properties
Questions?

http://caperl.links.org/

(not up yet)