The Open Kernel Environment
- spinning Linux -

Herbert Bos
Bart Samwel
{herbertb,bsamwel}@liacs.nl
Outline

- Goals + motivation
- Architecture
  - code loader
  - compiler
  - access to kernel
- POC
- Current work
- Conclusion
What do we want to do?

- allow *unprivileged* users to load *native* code in Linux kernel in a *safe* manner
  - security
  - resource control
  - crashes

- Why? Performance
  - network monitoring/filtering
  - active networks
Components

- Safe code loader
  - "who is allowed to load what modules?"
- Extensible compiler
  - "which programmer is allowed to do what?"
- Accessing kernel resources safely
  - "how do we make sure the kernel’s integrity is preserved?"
- Make rules for extensible compiler
  - *Legislator*
SPIN

- Extensible OS
  - "protection is a language issue"
- allows modules to be loaded in the kernel
- extensibility based on exported interfaces
- based on safety properties of Modula-3
  - type safety, auto storage mgmt, interfaces
SPIN

- kernel resources referenced by *capabilities*
  - pointer to block of mem, type declared in I/F
  - cannot be used inconsistent with type
  - unforgeable, statically checked
- Protection domain: set of accessible names
  - normally: VM
  - SPIN: explicit namespaces supported by language
- Code signed by Modula-3 compiler
The OKE

- non-privileged users load *native* code in krnl
  - *make resources available depending on role*
  - *many different privileges/roles*
  - *not: separate language for each class of users*
- don’t rely on interpreted language
- "elastic" compiler/language
  - *compiler applies extra rules depending on authorisation*
  - *so: language is customised based on credentials*
OKE Code Loader

- extend `insmod/rmmod`
- CL security policy based on trust mgmt
  - CL has set of *policies*
  - accepts blob of code + set of *credentials*
  - checks credentials
    - match: load module
    - no match: reject
  - implemented using *KeyNote*
OKE Code Loader

"who is allowed to load what?"

"who is allowed to load what?"
What are we allowed to do?

- "why should safe code not be allowed to run in the kernel?"
  - we want to restrict access to resources
  - limit number of modules, amount of memory, number of cycles, etc.
  - dependent on users!
- yet another safe language for specific purpose?
  - restricts towards lowest common denominator
- instead: one language that is restricted depending on role
Compiler: elastic languages

- depending on who the code-loading party is
  - more or less access is given to
    - language constructs
    - resources (incl. kernel functions + data structures)
  - extra checks or functions may be added

<table>
<thead>
<tr>
<th>trivial example: looping</th>
<th>other example: cycle counting</th>
</tr>
</thead>
<tbody>
<tr>
<td>no loops (students)</td>
<td>hard max (students)</td>
</tr>
<tr>
<td>limited loops (staff)</td>
<td>growing max (staff)</td>
</tr>
<tr>
<td>unrestricted loops (sysadmin)</td>
<td>don’t count (sysadmin)</td>
</tr>
</tbody>
</table>
Compiler: elastic languages

- many potential dangers can be spotted
  - at source level
  - after construction of AST
  - at intermediate code level

- elastic language = normal language ("C")
  - on which arbitrary restrictions can be placed
  - based on credentials
  - conceptually: separate languages (subsets)

- restrict at very early stage
Bygwyn compiler

- "you can’t always get what you want, but you get what you need"
- compiler is
  - Trusted
  - extensible (extra rules may customise language)
- customisation depends on authorisation
Bygwyn

- issues:
  - object code should be *tamperfree*
  - user's *authorisation* should be checked
- use *KeyNote* to convey and check authorisation at *coarse* granularity
  - "role" or user class
  - triggers compiler to load extra rules
Bygwyn compiler

"compile and sign"

"once loaded, who is allowed to do what?"

module C S

BYGWYN

my code

credentials

user

extra rules for user (given credentials and policies)
Access to kernel

- via interface (similar to SPIN)
- wrapped on the module code by compiler
- exceptions
The Legislator

- Who creates these extra rules?
- Legislators
  - customisation tools for target language
  - example: the *OKE legislator*
    - GUI based
    - tick boxes, fill in forms
    - no knowledge of compiler internals needed
Simple prototype

- Pascal (friendly grammar)
- customise almost anything
  - e.g. disallow pointers, restrict pointers, array bounds checks, etc.
- give safe access to kernel
  - packet filtering example
  - between 3 and 5 times faster than BPF
- often leads to toy languages
  - and/or need lot of runtime checks
Example: array bound checking

```c
char a[N];
a[i] = x;
```

- **option 1: don’t check**
- **option 2: check thoroughly**
  - ☀️CMP, JMP
  - ☀️terminate (e.g. throw exception)
- **option 3: fast check**
  - ☀️make sure index is always in range: a[i%N]
  - ☀️errors are programmer's problem
  - ☀️% is slow
  - ☀️boundary at power of 2 and chop off high bits: 1 AND

which check is used depends on the user’s credentials and/or preferences
Tough problems

- Namespace
- Pointers
  - NULL dereferences
  - dangling pointers
  - pointer arithmetic, etc.
- Sharing datastructures with the kernel
  - who frees the data?
- Recursion
Tough problems

- How can we interrupt a module?
  - what if module just obtained a *kernel lock*?
  - what if *deadlock* occurs?
  - what if it leaves kernel in *inconsistent* state?

As for the latter issue:

- if there is danger of that, module should not have received the authorisation to do so
  - encapsulate better
  - for certain users: provide timeouts on locks, etc.
We really want something like C

- so we now use modified "Cyclone"

- Crash-free "C"
  - different types of pointer
  - runtime checks (but only where needed)
  - region based protection

- however, we needed more (+ less)
  - share memory with kernel
  - interrupt modules
  - CPU usage
  - extensibility
Conclusions

- simple prototype is a dead end
- unprivileged users are allowed to load code in kernel
  - can be very fast
  - very much like C
- Cyclone is very good
- but needs many changes
- much work to be done