Trusted Linux:
A Secure Platform for Hosting Compartmented Applications

Tse Huong Choo
Hewlett Packard Laboratories 2001

Trusted Linux: Contents

- **Usage Scenario & Major Features**
- **Kernel Changes Explained**
- **Access Control Checks**
- **Filesystem Protection Mechanisms**
- **Application Integration - Legacy Java App**
  - Approach Taken
  - Various Configurations Explained
- **Summary**

Trusted Linux - Uses

Usage Scenario

- Internet Gateway Systems
- Secure Platform for Hosting Multiple Network Services

Examples

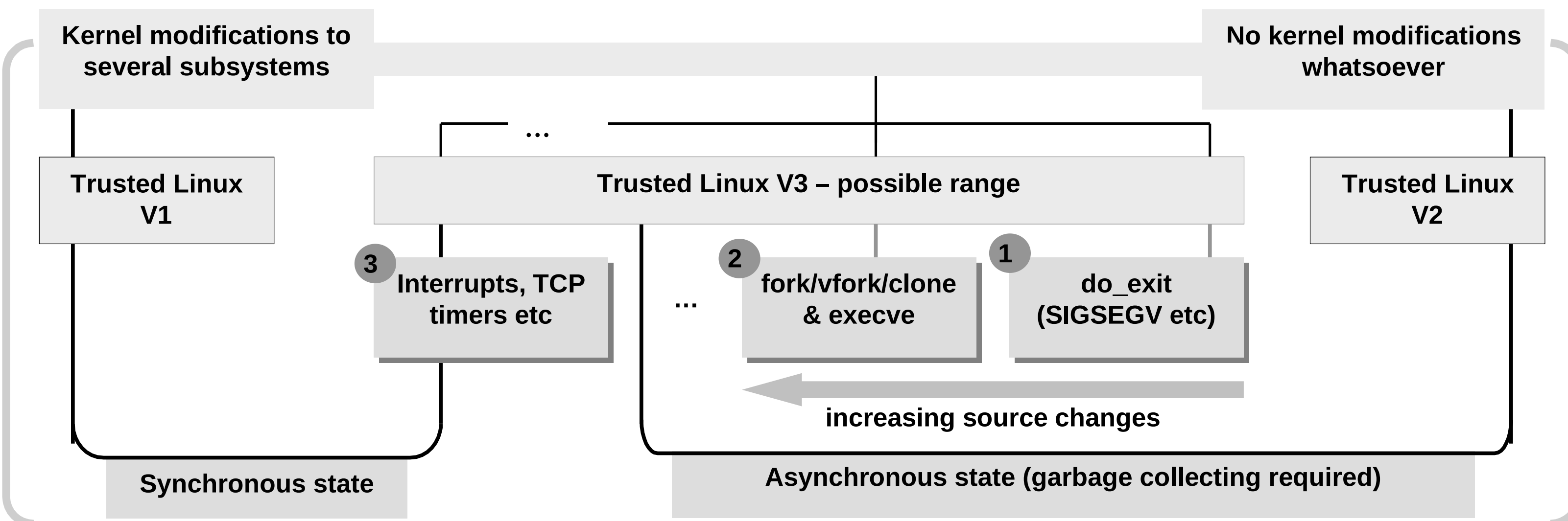
- Front-end Web server farms
- HTTP-fronted legacy applications residing on back-end servers
- Classic INSIDE/OUTSIDE configurations

Platform Support

- Based on Linux for IA-32, optimised for SMP-capable systems
- Mandatory Security properties
- Binary compatibility with existing software
- Pre-packaged compartment-aware applications provided

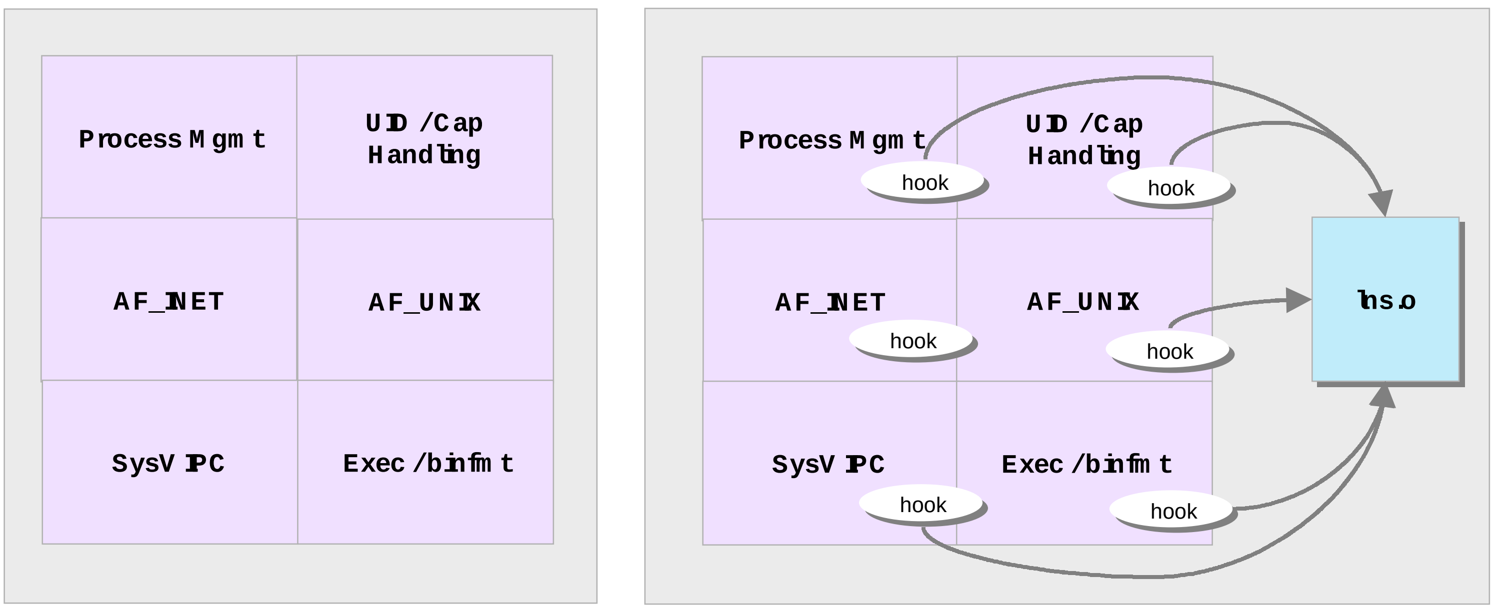
Building Containment

- **Options available for containment OS**
  - Tradeoff between direct kernel changes/performance and maintainability
  - In the end, kernel changes proved relatively minor



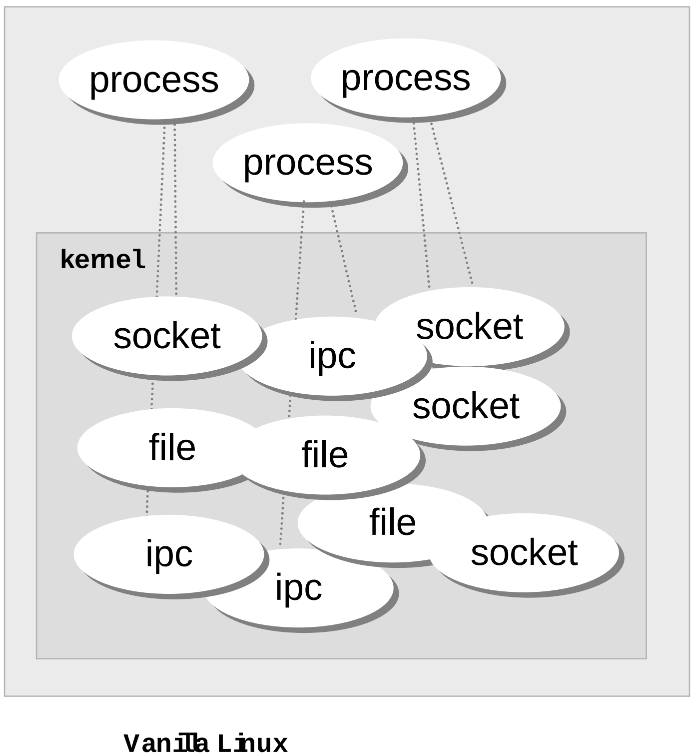
Major Features

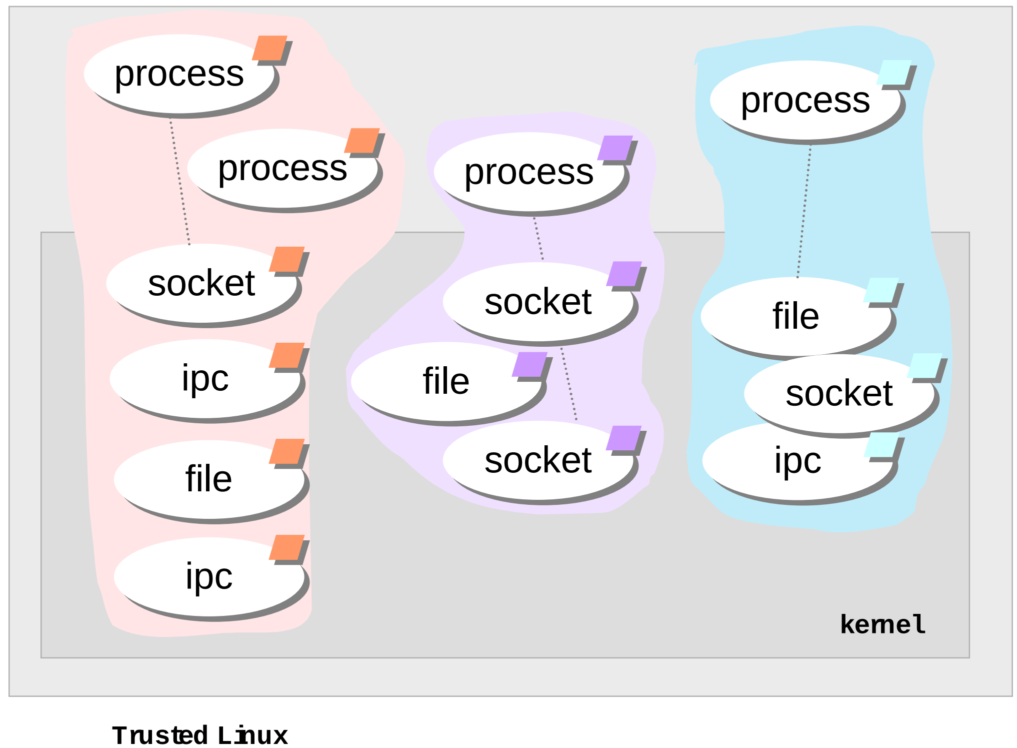
- **Kernel changes fall into 2 categories:**
  - **hooks** - access-control callouts placed thoughout kernel
  - **lns.o** - kernel module which implements the decision function into which the hooks call into



Effect of Changes

- **Kernel resources (individual variables) are tagged**





Trusted Linux New Datatype

- **Custom datatype to hold tag**
  - tag is a 32-bit scalar value (unsigned long on i386)
  - tags can be copied around without resource management
  - tags are cheap to use / destroy

```
#define CSI_INVALID_SL      0x00000000

typedef struct _csecinfo {
    unsigned long  sl;
} csecinfo;
```

Notes
Default initialisation value
Enclosing struct allows room for expansion

- **Tag applied to various kernel datatypes**
  - struct socket, struct sock, struct task_struct, struct sk_buff, etc.

Example Modification (struct sk_buff)

- **A struct sk_buff used throughout networking code**
  - In IPV4, it represents an individual IP packet (or fragment)
  - In UNIX domain sockets, it is an individual message buffer
  - In NIC device drivers, sk_buffs represent an entire frame

```
struct sk_buff {
        /* These two members must be first. */
        struct sk_buff  * next;                /* Next buffer in list               */
        struct sk_buff  * prev;                /* Previous buffer in list           */

        struct sk_buff_head * list;            /* List we are on                    */
        struct sock     *sk;                   /* Socket we are owned by            */
        struct timeval  stamp;                 /* Time we arrived                   */
...
#ifdef CONFIG_NET_SCHED
        __u32           tc_index;
#endif

#ifdef  CASPER
        csecinfo        csi;
#endif  /* CASPER */

};
```

Example Modification (skb_clone)

- **skb_clone() serves to make copies of sk_buffs**
  - tag needs to be propagated when sk_buffs are cloned

```
/**
 *      skb_clone       -       duplicate an sk_buff
 *      @skb: buffer to clone
 *      @gfp_mask: allocation priority

struct sk_buff *skb_clone(struct sk_buff *skb, int gfp_mask)
{
        struct sk_buff *n;

        n = skb_head_from_pool();
        if (!n) {
                n = kmem_cache_alloc(skbuff_head_cache, gfp_mask);
                if (!n)
                        return NULL;
        }
        ...
#ifdef CASPER
        n->csi = skb->csi;
#endif
        return n;
}
```

Typical Access Control Check

- **Packet delivery (TCP)**
  - Single decision function - cnet_lookup_rule()

```
#ifdef CASPER
int tcp_v4_do_rcv(struct sock *sk, struct sk_buff *skb)
{

    ...

    if( !cnet_lookup_rule( &skb->csi, &sk->csi, RTC_TCP_RCV, skb ))
    {
        if( skb )
            kfree_skb(skb);
        return 0;
    }
    return tcp_v4_do_rcv_stub(sk, skb );
}
#endif
```

csecinfo of source

csecinfo of destination

Context under which check is performed

Additional data from context

Programming Idioms

Datatype splicing is common

- memcpy() of portions of structs across different types busts type-checking
- Example:

struct tcp_tw_bucket (TIME_WAIT bucket for closing sockets)
struct sock (socket representation in kernel)

Both types share common members at the front of each struct

```
void some_function( …, struct sock *sk, … ) {
        struct tcp_tw_bucket *tw = (struct tcp_tw_bucket) sk;
```

!

Lifetime of variables must be tracked

- Ensure CSI_INVALID_SL is assigned when variables initially created
- Variable deallocation is non-issue: scalar tags need not be deallocated
- Examples:

struct sk_buff - alloc_skb() / sk_buff.c
struct sock - sk_alloc() / sock.c

Filesystem Protection

- **Desired semantics**
  - Operates on a per-compartment basis
  - Makes rest of filesystem inaccessible outside allocated portion, in case chroot fails
- **Managebility**
  - Per-file specifications are too bulky
  - Some precedence ordering required to secure ‘default’ cases
- **Focus on typical application behaviour & requirements**
  - Application specific logfiles typically reside in a subdirectory
  - Configuration files are read-only, often clustered together
  - Some form of content overwrite protection whilst allowing content update from another compartment

Filesystem Protection Mechanisms

- **Rules specifying access-control on a per-compartment basis**
  - Format: **[COMPARTMENT] [PATH] [ATTR-BITS]**
  - e.g

	COMPARTMENT	PATH	ATTR-BITS
1	WEB1	/compt/WEB1/apache/logs	append
2	WEB1	/compt/WEB1/apache/htdocs	readonly
3	WEB1	/compt/WEB1/	read-write
4	WEB1	/	no-access

Meaning:

1. Append-only to access_log, error_log ssl_* logfiles
2. Content protection against attempted overwrite via buffer overflow
3. General access within the allocated space for this compartment
4. No access whatsoever outside /compt/WEB1

Example Application Integration - 1

- **Assume legacy Java Servlet/JSP application – JAPP**
  - Java Authorisation Server using RMI
- **Approach taken:**
  - Hide from external probing as many components as possible
  - Factor out as much potentially untrusted code as possible from direct external access
  - Separate groups of components into clearly defined process & communication boundaries
  - Hide sensitive application-configuration files from public access
  - Prevent the hosted services from accessing each others configuration files and those of the JAPP components
  - No source-code changes to JAPP if possible

Example Application Integration - 2

Standard Configuration

- This is the configuration one might use on a non-compartment system e.g. on standard Linux. All processes and RMI objects are potentially directly accessible.
- There is litte protection against a buffer-overflow attack in the HTTP-server leading to root-equivalent access.

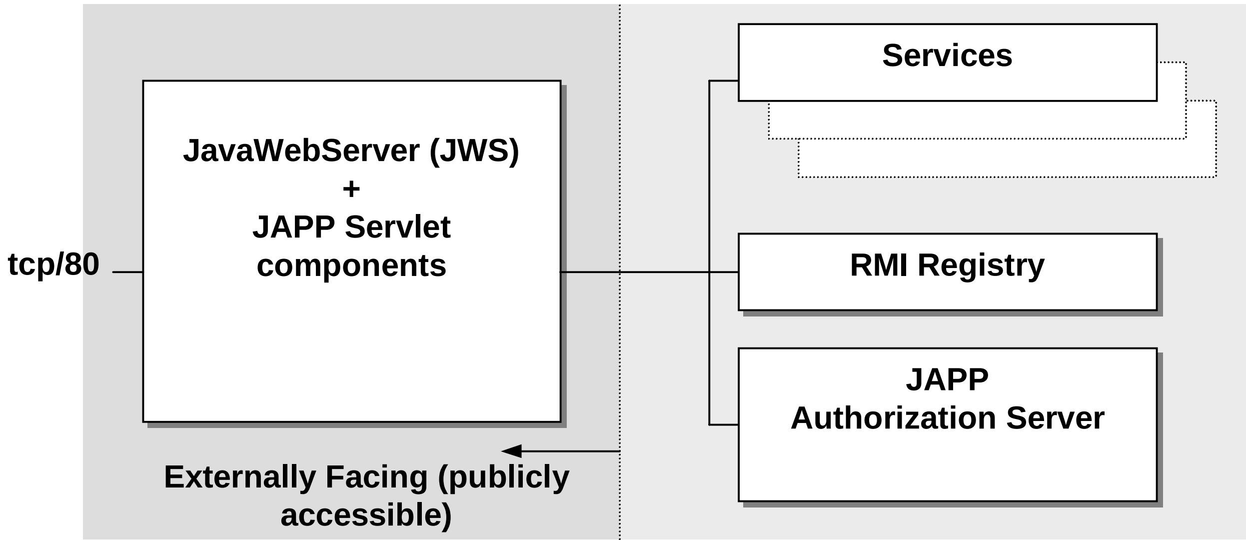
	Hidden from External Access	Levels of Indirection from direct access	Component present in external compartment
HTTP-Server	No	0 (direct)	Yes*
JAPP Servlet	No	0	Yes*
RMI Registry	No	0	Yes*
JAPP Server Objects	No	0	Yes*
Services Individually Separate		No	
HTTP-Server can gain root		Yes	

*: Trivially true, since the entire system is considered a single compartment

Example Application Integration – 3a

3+X Configuration

- Configuration is straightforward
- Single rule allowing access via tcp/80 for remote clients
- Rules connecting
  - Java Web server + Servlets
  - RMI Registry
  - JAPP Authorisation Server

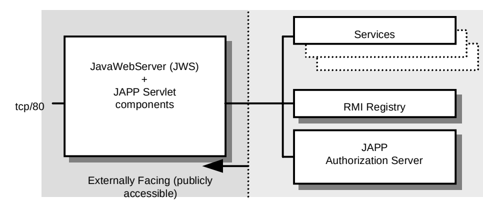


Example Application Integration – 3b

3+X Configuration

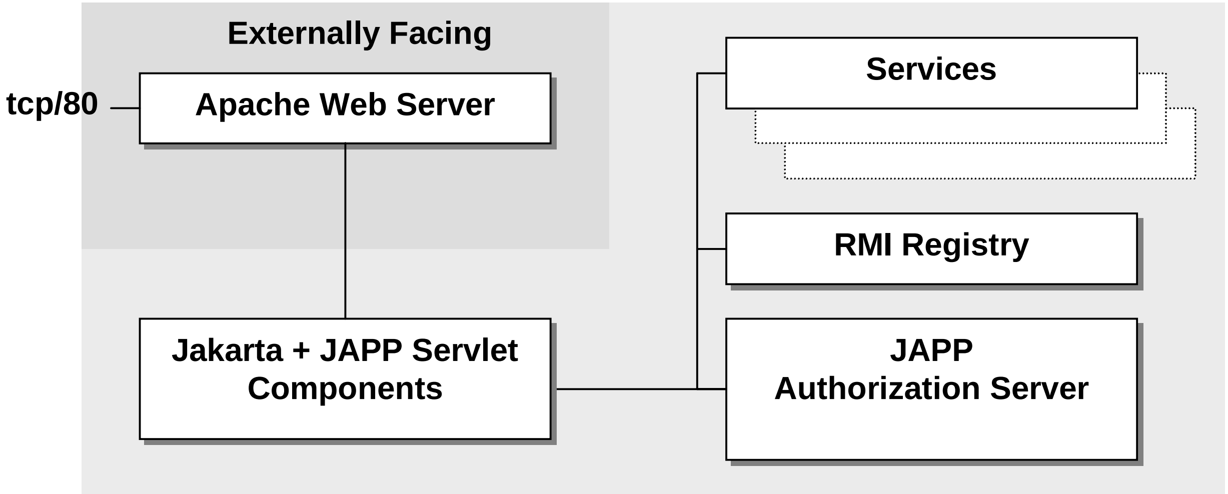
- Advantage of hiding the connection points of the RMI server-objects from external access
- Externally-facing compartment also sealed against root transitions as a general precaution
- Avoid possiblity of getting root in externally-facing compartments

	Hidden from External Access	Levels of Indirection from direct access	Component present in external compartment
HTTP-Server	No	0 (direct)	Yes
JAPP Servlet	No	0 (direct)	Yes
RMI Registry	**Yes**	1	**No**
JAPP Server Objects	**Yes**	1	**No**
Services Individually Separate		**Yes**	
HTTP-Server can gain root		**No**	



Example Application Integration – 4a

- **4+X Configuration - Tighten previous configuration**
  - Observation: too much potentially untrusted code resides in one externally-facing compartment
  - Possible to factor out more code from the single external compartment into another separate internal compartments
  - Use out-of-process servlet container, connected by single rule

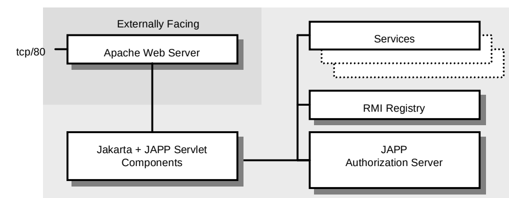


Example Application Integration – 4b

4+X Configuration

- Web-server compartment running substantially fewer pieces of code
- Note that the back-end RMI objects are now twice-removed from external-access where previously only once-removed

	Hidden from External Access	Levels of Indirection from direct access	Component present in external compartment
HTTP-Server	No	0 (direct)	Yes
JAPP Servlet	**Yes**	**1**	**No**
RMI Registry	Yes	**2**	No
JAPP Server Objects	Yes	**2**	No
Services Individually Separate		Yes	
HTTP-Server can gain root		No	



Sum m ary

- **Principle Uses**
  - Secure gateway systems
  - Web-fronted applications requiring access to relatively unprotected back-end servers
- **Platform Support**
  - Targeted at Linux IA-32 SMP systems
  - Layered installation on top of well known distributions
  - Binary compatibility whilst gaining additional mandatory security properties
- **Application Integration Using Trusted Linux**
  - Factor out as much code that is directly accessible
  - Define communications boundaries
  - Define interfaces as narrowly as possible
  - Use chroot/restricted filesystems to reduce accessibility of configuration files