Kernel Level Vulnerabilities

Behind the Scenes of the 5th Argus Hacking Challenge

Chaos Communication Congress 2001

Originally presented on Black Hat Briefings 2001

Last Stage of Delirium Research Group

http://LSD-PL.NET

contact@lsd-pl.net

The Last Stage of Delirium Research Group

- The non-profit organization, established in 1996,
- research activity conducted as the LSD is not associated with any commercial company,
- four official members,
- all graduates (M.Sc.) of Computer Science from the Poznań University of Technology, Poland
- for the last six years we have been working as Security Team at Poznań Supercomputing and Networking Center.

Our fields of activity

- Continuous search for new vulnerabilities as well as general attack techniques,
- analysis of available security solutions and general defense methodologies,
- development of various tools for reverse engineering and penetration tests,
- experiments with distributed host-based Intrusion Detection Systems with active protection capabilities,
- other security related stuff.

Presentation overview

- General introduction
- Basics of Pitbull Foundation Intrusion Prevention System
- About 5th Argus Hacking Challenge
- The ldt kernel level vulnerability
- The specified phases of the attack against system with security enhanced by Pitbull Foundation
- Technical details of successful proof of concept code
- Summary and final remarks

Motivations

- The main goal of this demonstration is to present the technical details of successful exploiting kernel level vulnerabilities
- The special emphasis will be put on potential consequences of such errors
- The Argus Hacking Challenges as an interesting case study of such consequences
- There is no such thing like 100% secured system
- The system can be considered secure only in the specific place at the specific moment

Pitbull Foundation

Intrusion Prevention System

- Software enhancement to the operating system that is based on the Trusted Operating Systems (TOS) technology
- It got B1 security evaluation under the Information Technology Security Evaluation Criteria (ITSEC)
- It was approved for governmental use by NSA and DoD
- It has never been hacked before...

Pitbull Foundation features

- Removal of superuser privileges
- Least privilege
- Information compartmentalization and Mandatory Access Control (MAC)
- Role compartmentalization
- Kernel-level enforcement

Privileges

- They are attributes of a process or file system object that define what security relevant actions a specific code is allowed to perform
- In a standard UNIX OS root inhibits all power in the system
- In Pitbull root privileges are divided into many sub-privileges (**PV_DAC_R**, **PV_FS_MOUNT**,...)
- There are new privileges added to the system (**PV_MAC_W**, **PV_PROC**,...)

The least privilege principle

- A process or executable should only have the minimum necessary privileges needed for the performance of its tasks
- A user should only have the authorizations which are required for the performance of his duties
- Privileges are enabled or disabled around the smallest section of code that requires them

Information Compartmentalization

- Standard Unix operating systems implement access control with the use of Discretionary Access Control (DAC): permission bits and ACLs
- Mandatory Access Control (MAC) is a mechanism providing information compartmentalization in TOS
- Unlike standard DAC, MAC restrictions cannot be overruled by a root-owned process

Information Compartmentalization (2)

- Every object on the system, including both files and processes, has a sensitivity label (SL)
- There are two components of a sensitivity label: *classification* and *compartments*
- Classification is a hierarchical component of SL. It defines sensitivity of information
- Compartments are non hierarchical components of sensitivity labels. They define sets of information categories

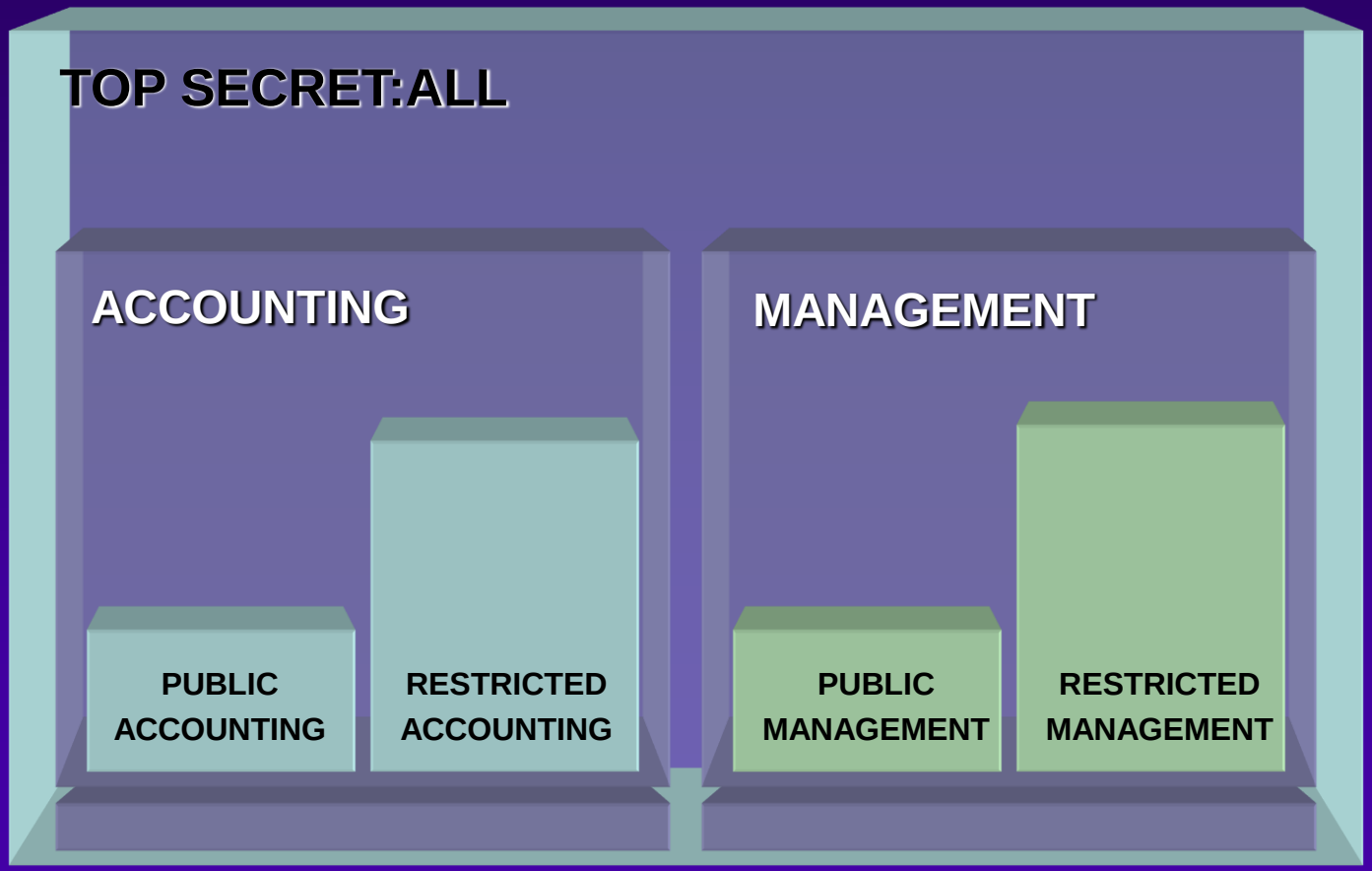
Information Compartmentalization (3)

CONFIDENTIAL

RESTRICTED

PUBLIC

IMPL. LOW



Information Compartmentalization (4)

- A process cannot open a file for reading unless the SL of the process dominates the SL of the file
- A process cannot open a file for writing unless the SL of the process equals the SL of the file
- Unless a process has the privilege needed to change an SL, the process cannot change its own SL or the SL of any process or file on the system

Role Compartmentalization (Authorizations)

- A combination of privileges and authorizations is used to enforce role compartmentalization
- Authorizations are assigned to users (privileges to processes). They are required in order to use privileges
- If a user is not authorized for a privilege, he will not be able to use that privilege even if he is permitted to execute a privileged file

Kernel Level Enforcement

- Security decisions are made at the operating system kernel level
- The security decisions are made as close as possible to the resources being protected
- The system is more secure than any combination of user-level or application-level security
- The standard UID 0 checks are replaced with more specific and more targeted privilege checks
- Root exploits are no longer useful to intruders

LSD comments on Pitbull Foundation

+ Provides additional kernel level protection
+ It is based on TOS technology (proved mathematical security models)
+ Protects against classical user level attacks

but...

- It is potentially open to new threats
- Some of its features seem to be overused
- Tight configuration of the system is very difficult. It requires a lot of experience and work.

5th Argus Hacking Challenge

Coincided with Infosecurity Europe 2001 Exhibition, held in London, 20-25 April 2001.

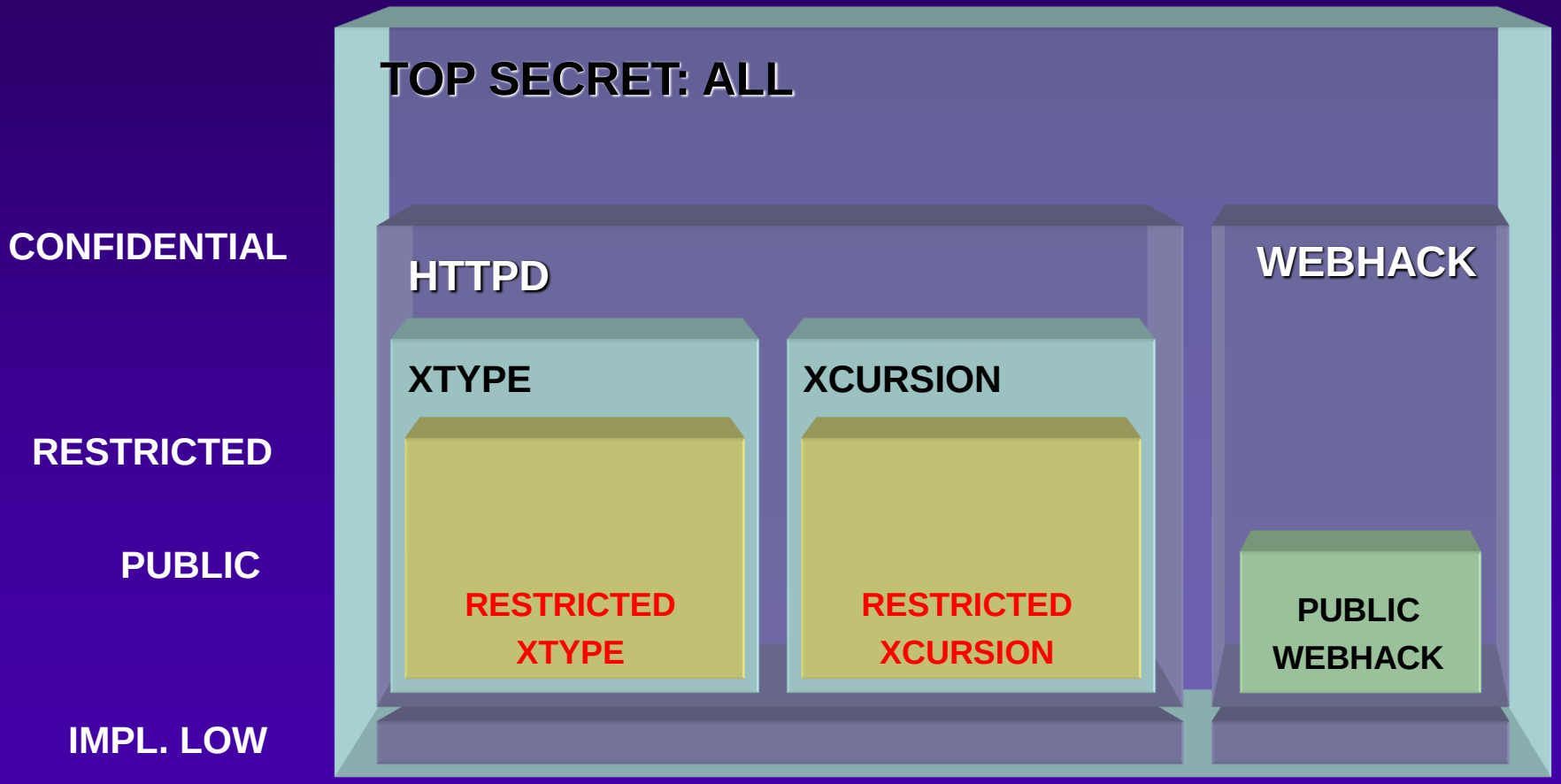
The rules:

- Hack the system secured with Pitbull Foundation 3.0 MU4 and servicing the web pages of two fictional companies: *Xtype* and *XCursion*
- Do it within 5 days time
- In a case of success reveal how it was achieved
- Get the prize of 50 000 USD!

Challenge system configuration

- Solaris 7 x86 with Pitbull Foundation 3.0 and .comPack (web protection) installed
- Partially secured OS (patches applied, the number of services decreased, *set-user-id* bits removed from many system binaries)
- Remote shell access via TSSH service to the public *webhack* account
- Separate and disjoint compartment definitions for user *webhack*, *httpd* server, *xtype* and *xcursion* web pages directories
- ASN rules for network protection

Simplified challenge system configuration (2)

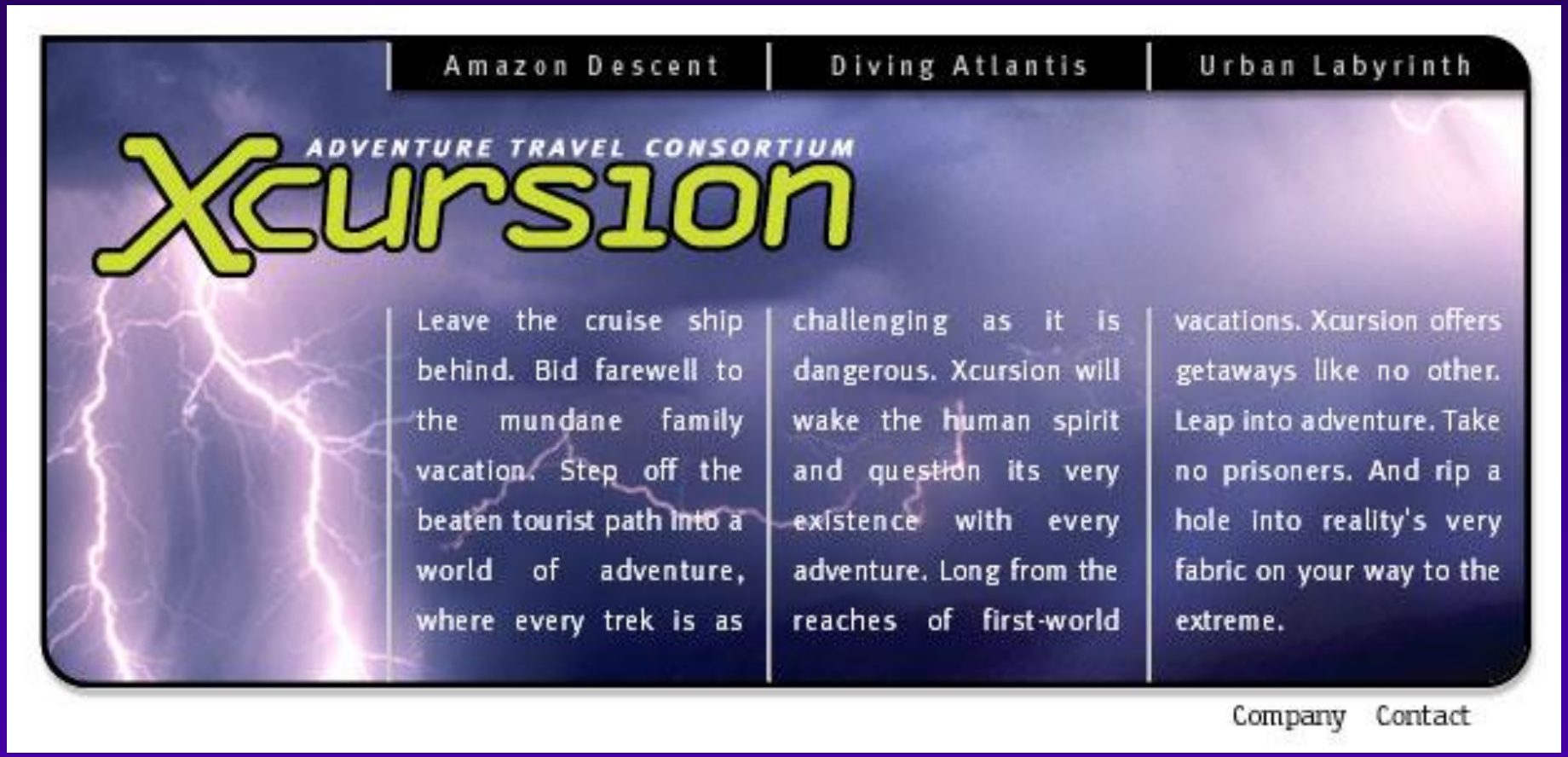


Challenge system configuration (3)

XTYPE

The companies and products depicted on these pages are ficitious and were created solely for the purposes of the Argus hacking contest. No resemblance to real companies, services, or products is intended.

Challenge system configuration (4) XCURSION



The companies and products depicted on these pages are ficitious and were created solely for the purposes of the Argus hacking contest. No resemblance to real companies, services, or products is intended.

The weapon: ldt bug

- Specific to x86 architecture and OS protection mechanisms provided by x86 family of processors
- Kernel level vulnerability that allows user mode processes to install call gates in their Local Descriptor Table
- Installed call gate could be an entry point to 0 protection level, thus it would allow code execution at the OS kernel level
- Uncommon and tricky to exploit

ldt bug (2) - destroying the myth

- First reported in a NetBSD Security Advisory in January 2001 (reported by Bill Sommerfeld)
- According to the advisory the following operating systems were vulnerable: Solaris, NetBSD / OpenBSD
- We found its existence in SCO Unixware and SCO OpenServer
- At the time of a challenge Solaris x86 as well as Pitbull were still vulnerable (!)
- Is it a Solaris or Pitbull's bug ?

Attack phases

- The attack was performed within 24 hours time and it consisted of several phases, reflecting major modifications done to the ldt proof of concept code
- During each of the phase some new idea was tried
- There were some mistakes done, that fortunately did not lead to the system reboot (could be treated as an action against the Challenge rules)

Phase 0 (09:00 CET, Friday)

- Finding information about the Challenge on a security news portal
- Installation of Pitbull 3.0 MU4 on a local machine for the “know your enemy” purposes
- System configuration and tuning in order to reflect the challenge conditions
- Verification of the ldt bug on the testbed system
- Development of fully operational proof of concept code for ldt bug

Phase 1 (18:00 CET, Friday)

- Login to the webhack account through SSH
- Verification of user webhack privileges
- Verification of Pitbull settings for user webhack
- General look around the system
  - local vulnerabilities
  - ASN network protection
- Verification of the ldt bug on a challenge system

Phase 1: Initial Privileges

Subject:
`LSD hunter process`

uid: webhack
sl: PUBLIC WEBHACK
privs: none

Access checks: **READ / WRITE**

MAC: compare SUB SL with OBJ SL

WEBHACK ?? XTYPE
compartments disjoint

READ FAILED WRITE FAILED

DAC: none

File system object:
`/www/xtype/htdocs/index.html`

MAC

Sensitivity label

Classification	*Compartments*
RESTRICTED	**XTYPE**

DAC

Rights mask

Owner			*Group*			*Other*		
R	W		R			R		

owner: **xtype**

group: **xtype**

Phase 2 (20:00 CET, Friday)

- Application of Solaris x86 ldt proof of concept code to the challenge system
- Gaining standard root user privileges (uid=0)
- Playing with the new set of privileges (root has no power in the system)

Phase 2: Gaining standard root user privileges (uid=0)

Subject:
`LSD hunter process`

uid: root ✓
sl: PUBLIC WEBHACK
privs: none

Access checks: **READ / WRITE**

MAC: compare SUB SL with OBJ SL

WEBHACK ?? XTYPE
compartments disjoint

READ FAILED WRITE FAILED

DAC: none

File system object:
`/www/xtype/htdocs/index.html`

MAC

Sensitivity label

Classification	*Compartments*
RESTRICTED	XTYPE

DAC

Rights mask

Owner			*Group*			*Other*		
R	W		R			R		

owner: **xtype**

group: **xtype**

Phase 3 (21:00 CET, Friday)

- Preliminary attempts to bypass MAC protection
- Setting the *classification* component of the user webhack's effective SL to ”TOP SECRET”
- Getting the highest information access level in the WEBHACK compartment

Phase 3: Getting TOP SECRET classification

Subject:
`LSD hunter process`

uid: root ✓
sl: TOP SECRET ✓ **WEBHACK**
privs: none

Access checks: **READ / WRITE**

MAC: compare SUB SL with OBJ SL

WEBHACK ?? XTYPE
compartments disjoint

READ FAILED WRITE FAILED

DAC: none

File system object:
`/www/xtype/htdocs/index.html`

MAC

Sensitivity label

Classification	*Compartments*
RESTRICTED	**XTYPE**

DAC

Rights mask

Owner			*Group*			*Other*		
R	W		R			R		

owner: **xtype**

group: **xtype**

Phase 4 (22:30 CET, Friday)

- Setting the *classification* component of the user webhack's effective SL to "TOP SECRET"
- Setting the *compartments* component of the user's SL to ALL
- Obtaining the highest information access level in the protected system (read access to all its objects)
- Writing to target objects is denied

Phase 4: Getting TOP SECRET classification in ALL compartments

Subject:
`LSD hunter process`

uid: root ✓
sl: TOP SECRET ✓ ALL ✓
privs: none

Access checks: **READ / WRITE**

MAC: compare SUB SL with OBJ SL

TS ALL > RESTRICTED XTYPE
RESTRICTED XTYPE !> TS ALL
subject and object SL are not equivalent

READ ACCEPTED **WRITE FAILED**

DAC: compare uid with object owner
check right mask

root != xtype (users differ)
other[READ]=R

READ ACCEPTED

File system object:
`/www/xtype/htdocs/index.html`

MAC

Sensitivity label

Classification	*Compartments*
RESTRICTED	**XTYPE**

DAC

Rights mask

Owner			*Group*			*Other*		
R	W		R			R		

owner: **xtype**

group: **xtype**

Phase 5 (04:00 CET, Saturday)

- Setting the *classification* component of the user webhack's effective SL to "RESTRICTED"
- Setting the *compartments* component of the user's SL to XTYPE
- Obtaining read access level in the XTYPE compartment
- Writing to target objects is still denied

Phase 5: Getting RESTRICTED classification in XTYPE compartment

Subject:
`LSD hunter process`

uid: root ✓
sl: RESTRICTED ✓ XTYPE ✓
privs: none

Access checks: **READ / WRITE**

MAC: compare SUB SL with OBJ SL

RESTRICTED XTYPE = RESTRICTED XTYPE
subject and object SL are equivalent

READ ACCEPTED WRITE ACCEPTED

DAC: compare uid with object owner
check right mask

root != xtype (users differ)
other[READ]=R, other[WRITE] = 0

READ ACCEPTED WRITE FAILED

File system object:
`/www/xtype/htdocs/index.html`

MAC

Sensitivity label

Classification	*Compartments*
RESTRICTED	XTYPE

DAC

Rights mask

Owner			*Group*			*Other*		
R	W		R			R		

owner: **xtype**

group: **xtype**

Phase 6 (05:00 CET, Saturday)

- Setting the *classification* component of the user webhack's effective SL to "RESTRICTED"
- Setting the *compartments* component of the user's SL to XTYPE
- Setting *uid* of a process to *xtype* user
- Obtaining full access level (read and write) in the XTYPE compartment

Phase 6: Getting RESTRICTED classification in XTYPE compartment and uid=xtype

Subject:
`LSD hunter process`

uid: xtype ✓
sl: RESTRICTED ✓ XTYPE ✓
privs: none

Access checks: **READ / WRITE**

MAC: compare SUB SL with OBJ SL

RESTRICTED XTYPE = RESTRICTED XTYPE
subject and object SL are equivalent

READ ACCEPTED WRITE ACCEPTED

DAC: compare uid with object owner
check right mask

xtype = xtype (user is the owner)
other[READ]=R, owner[WRITE] = W

READ ACCEPTED WRITE ACCEPTED

File system object:
`/www/xtype/htdocs/index.html`

MAC

Sensitivity label

Classification	*Compartments*
RESTRICTED	XTYPE

DAC

Rights mask

Owner			*Group*			*Other*		
R	W		R			R		

owner: **xtype**

group: **xtype**

Phase 7 (07:00 CET, Saturday)

- Setting ALL process privileges (minimum, maximum, limited and effective sets)
- Obtaining full access level (read and write) to the protected system, regardless of the DAC and MAC settings
- Historically, this was the way how the Challenge system was hacked :-)

Phase 7: Setting all privileges for given process

Subject:
`LSD hunter process`

uid: root
sl: PUBLIC WEBHACK
privs: PV_ROOT* PV_SU* ✓

Access checks: **READ/WRITE**

MAC: checking for privileges
PV_ROOT* present

READ ACCEPTED WRITE ACCEPTED

DAC: checking for privileges
PV_ROOT* present

READ ACCEPTED WRITE ACCEPTED

File system object:
`/www/xtype/htdocs/index.html`

MAC

Sensitivity label

Classification	*Compartments*
RESTRICTED	XTYPE

DAC

Rights mask

Owner			*Group*			*Other*		
R	W		R			R		

owner: **xtype**

group: **xtype**

Defaced web page

XTYPE

ATTENTION!!!
We would like to inform you that what had to be done has been done. This site has been modified by Last Stage of Delirium (http://lsd-pl.net) during the Argus hacking contest. Thank you for your cooperation.

Defaced web page (2)

XCURSION

ATTENTION!!!
We would like to inform you that what had to be done has been done. This site has been modified by Last Stage of Delirium (http://lsd-pl.net) during the Argus hacking contest. Thank you for your cooperation.

Technical details

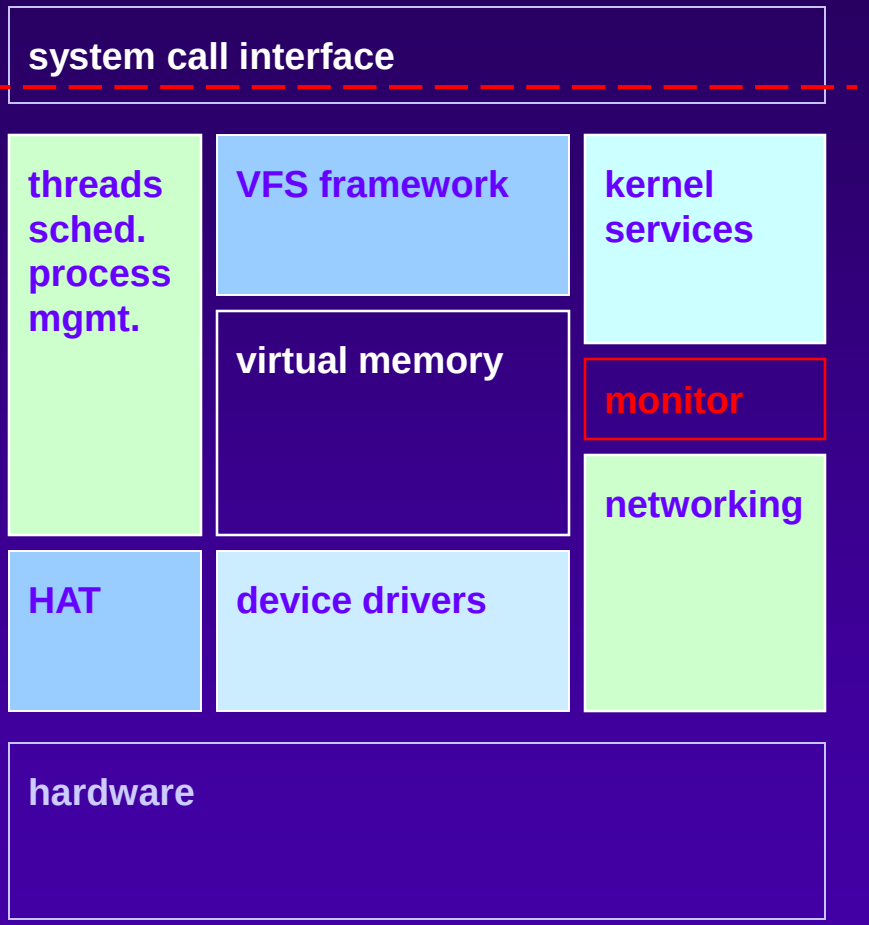
This part covers:

- kernel memory protection overview
- how is Pitbull integrated with Solaris x86 kernel ?
- what is x86 ldt bug ?
- why does this bug affect Pitbull product ?

... and finally:

- brief description of how to successfully exploit x86 **ldt** vulnerability in Solaris 7/8 x86 operating system enhanced with Argus Pitbull Foundation 3.0 MU4+ and Web Protection .comPack products

Pitbull protected kernel overview

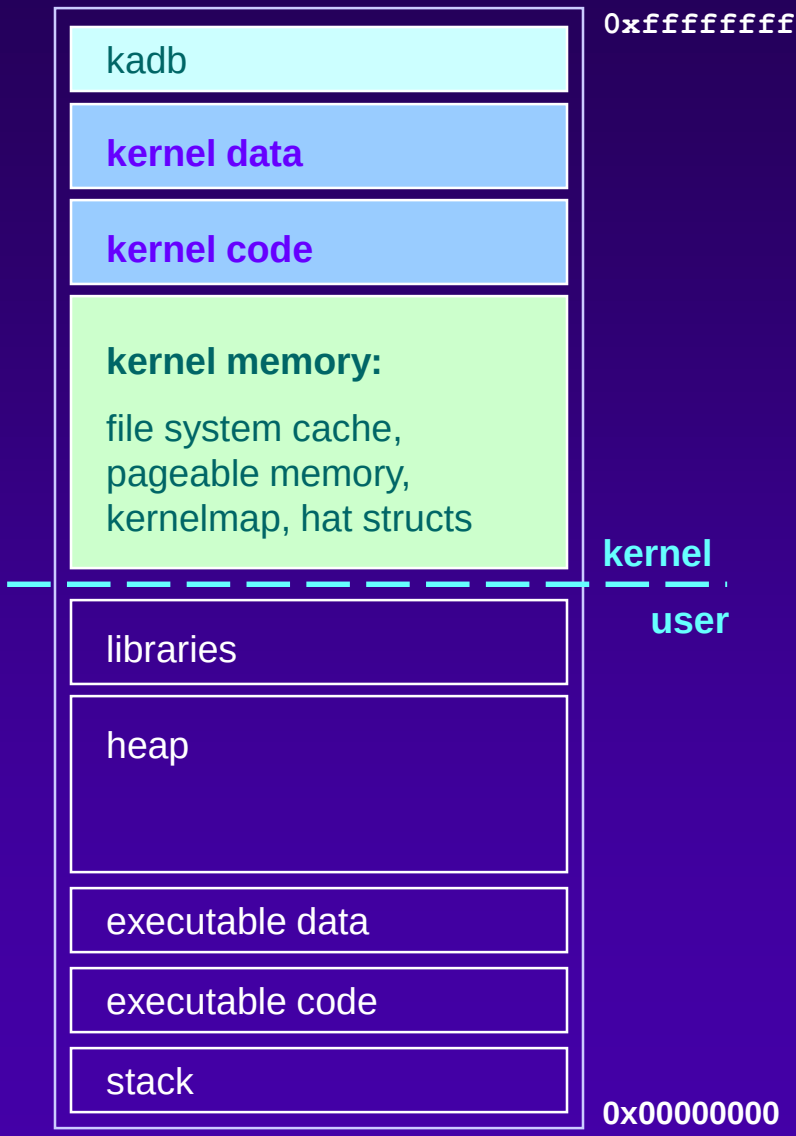


All critical system actions pass through Argus **Reference Monitor** Module

At least system call layer is intercepted

Theoretically Pitbull has a possibility to control and successfully block any operation initiated by user programs

Kernel virtual address maps

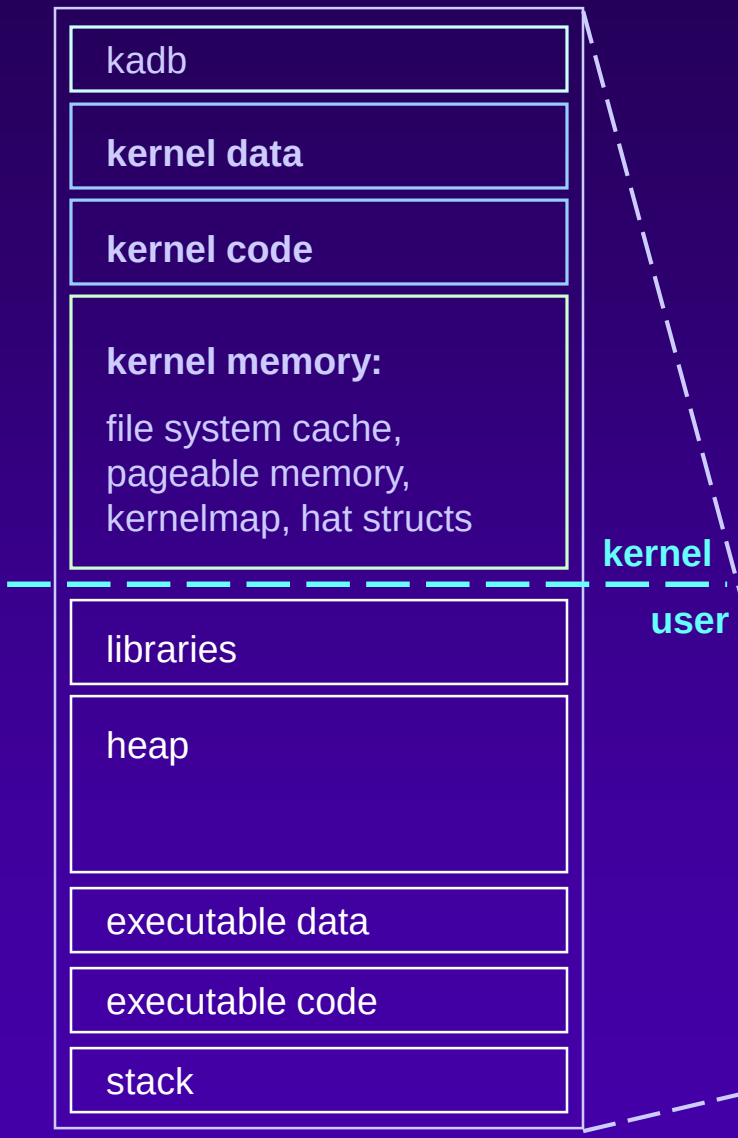


Process virtual address space is divided to kernel and user space

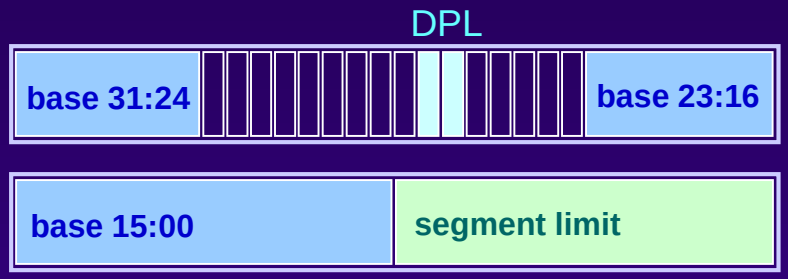
Advantages:

- No context switch occurs when transferring control from user to kernel space
- Kernel have direct access to whole user memory
- Kernel can easily distinguish to what memory space a given address belongs to

Memory protection



Code Segment Descriptors



In Solaris system, sensitive kernel data and code is protected on a **page level basis**

Process, while in kernel as well as in user mode, uses segment selectors that cover the whole 4GB virtual address space:

```
KCSSEL,KDSSEL:   RPL=0, DPL=0
UCSSEL,UDSSEL:   RPL=3, DPL=3
```

Processes accessing kernel services

To provide sufficient level of control while accessing code segments with different privilege levels, processors use special set of descriptors, called *gate descriptors*

There are four main types of such descriptors:

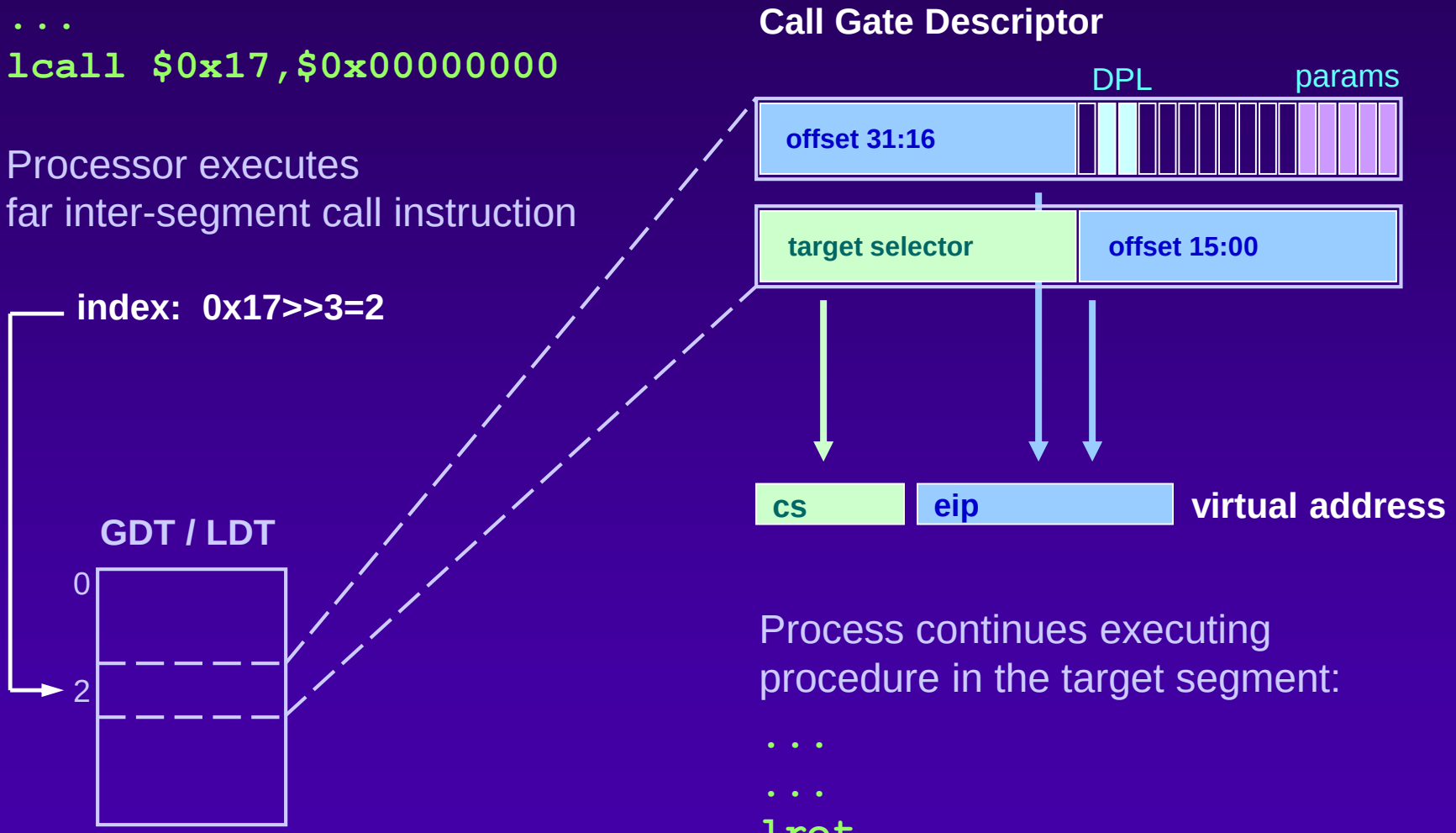
- *task gates* connected with task management
- *trap gates* for exceptions handling
- *interrupt and* ***call gates*** usually used to provide interface for accessing more privileged protection levels to user applications operating at the lower ones

x86 call gate mechanism

Process in user mode:

```
...
lcall $0x17,$0x00000000
```

Processor executes
far inter-segment call instruction



Process continues executing
procedure in the target segment:

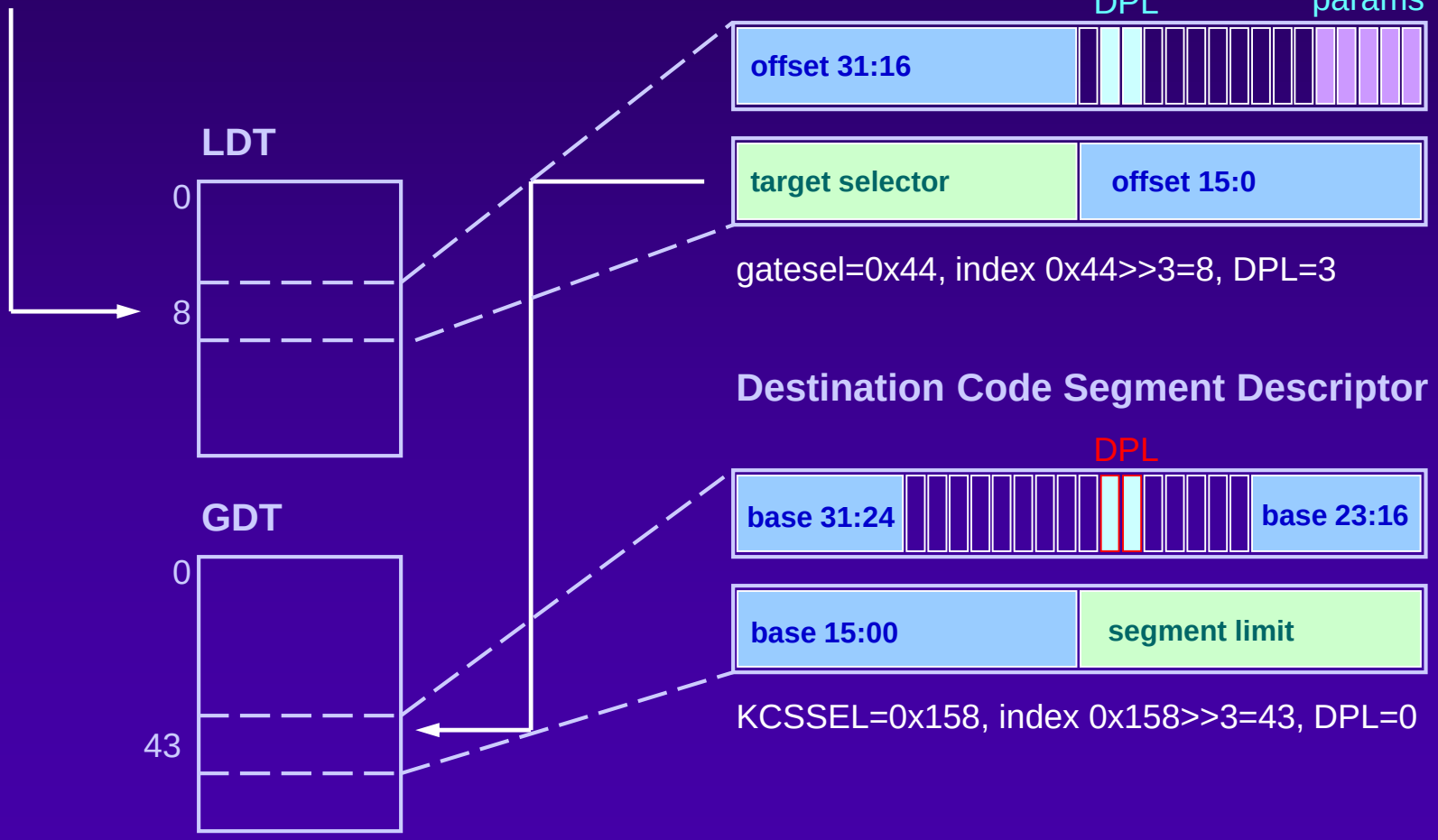
```
...
...
lret
```

ldt bug - sysi86() system call

DPL of destination segment descriptor is not checked

User program calls:

```
sysi86(SI86DSCR,struct ssd*)
```



Jumping to the kernel space

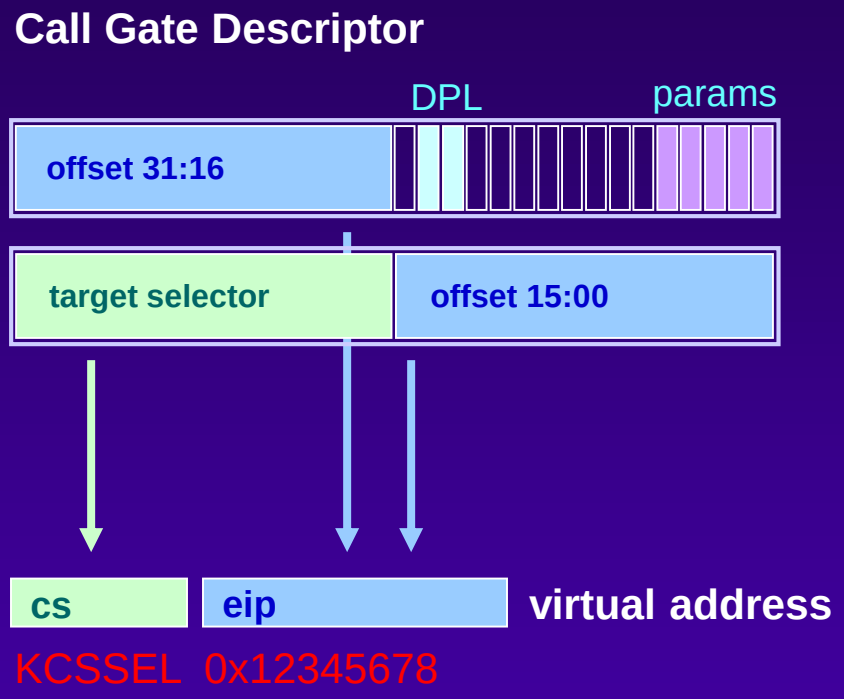
Installation of a call gate

```
s.bo=0x12345678;
s.sel=0x44;
s.ls=KCSSEL;
s.acc1=GATE_UACC|GATE_386CALL;
s.acc2=8;

sysi86(SI86DSCR,&s);
```

Call through the gate

```
lcall    $0x44,$0x00000000
```



Processor tries to execute instruction at new virtual address on most privileged protection level - **panic!**

Executing code on the kernel stack

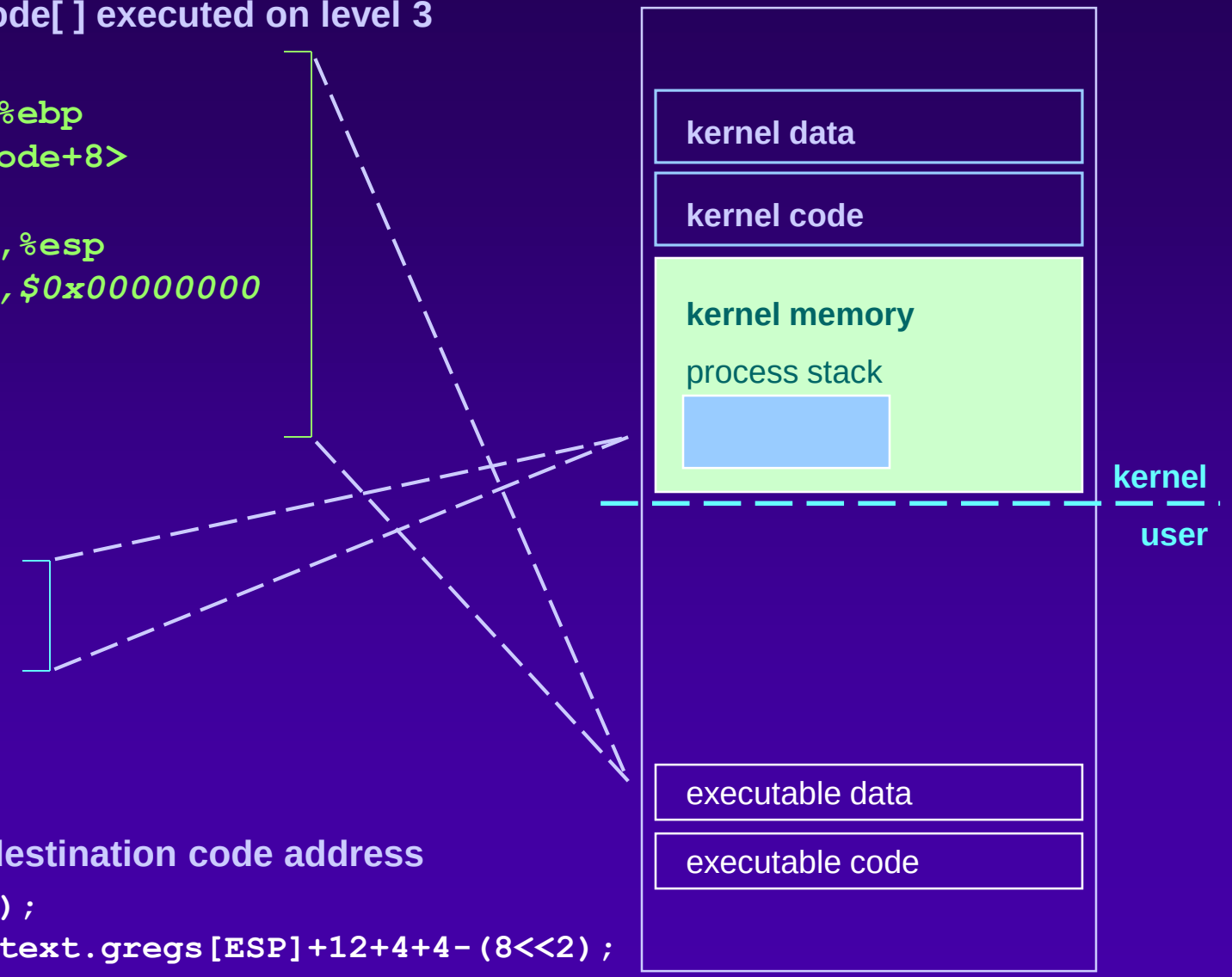
Portion of asmcode[ ] executed on level 3

```
pushl   %ebp
movl    %esp,%ebp
call    <asmcode+8>
popl    %esp
addl    $0x0d,%esp
lcall   $0x44,$0x00000000

leave
ret
```

and level 0

```
nop
nop
lret    $0x20
```



Calculating the destination code address

```
getcontext(&uc);
adr=uc.uc_mcontext.gregs[ESP]+12+4+4-(8<<2);
```

Executing code in user space

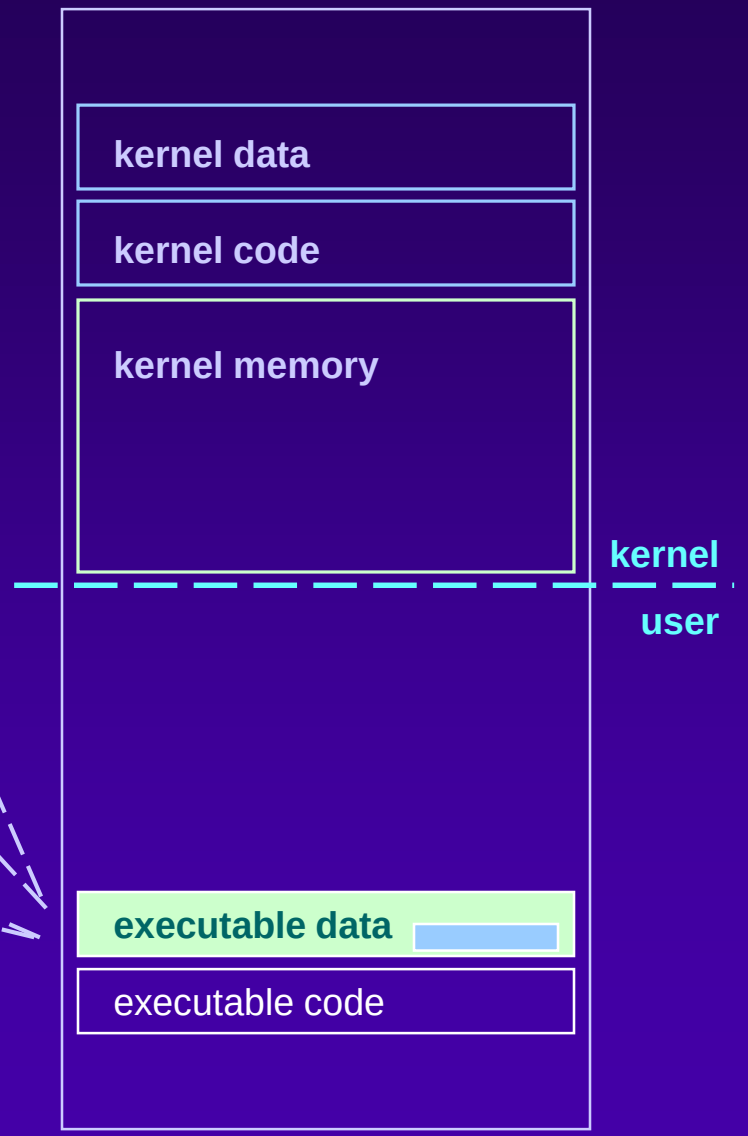
Portion of asmcode[ ] executed on level 3

```
pushl   %ebp
movl    %esp,%ebp
call    <asmcode+8>
popl    %esp
addl    $0x0d,%esp
lcall   $0x44,$0x00000000

leave
ret
```

and level 0

```
nop
nop
lret    $0x20
```



Destination code address is known

```
adr=&asmcode[21]
```

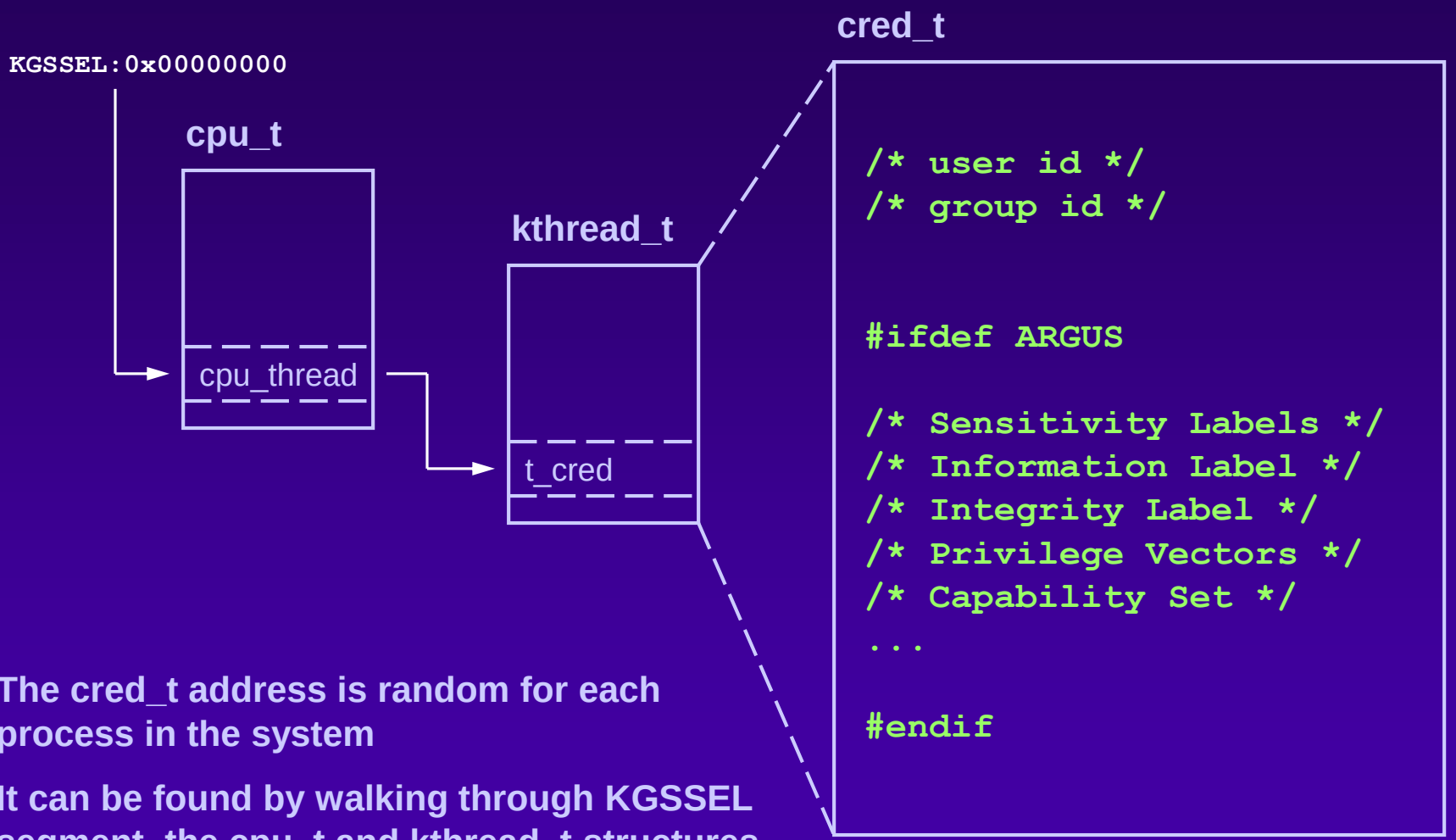
What can be done from the kernel level ?

There are a lot of possible actions that can be undertaken at the kernel level code:

- crash the system
- enable/disable Solaris or Pitbull security settings
- obtain raw access to disk devices
- leverage process credentials
- and more ...

At this point there are actually no active security protections left...

Finding process' cred_t on kernel heap

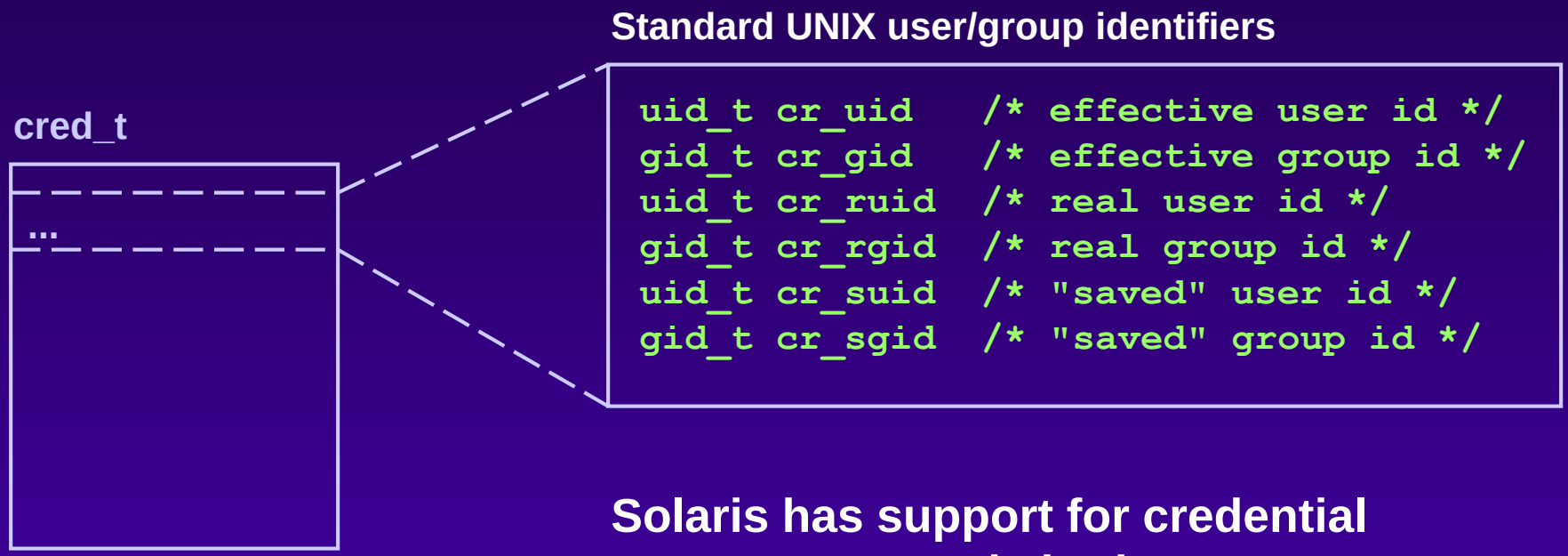


The cred_t address is random for each process in the system

It can be found by walking through KGSSEL segment, the cpu_t and kthread_t structures

```
ttoproc(curthread)-> p_cred
```

Modification: process user ID



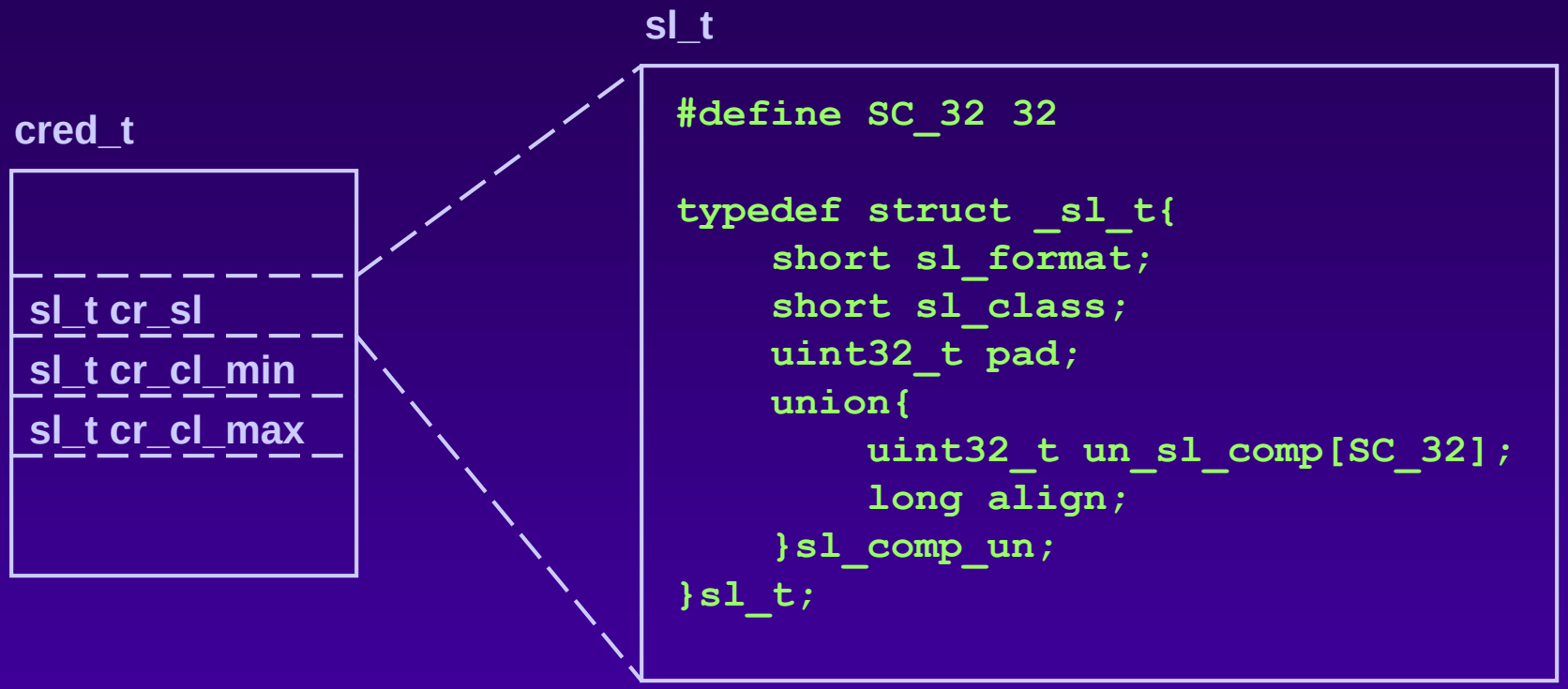
Solaris has support for credential management and sharing

- it is better to create copy of cred_t before changing it
- cr_ruid cannot be directly modified as it will result in credential index inconsistency (will crash the system)
- cr_uid is changed

```
setuid(getuid())

p_cred-> cr_uid
```

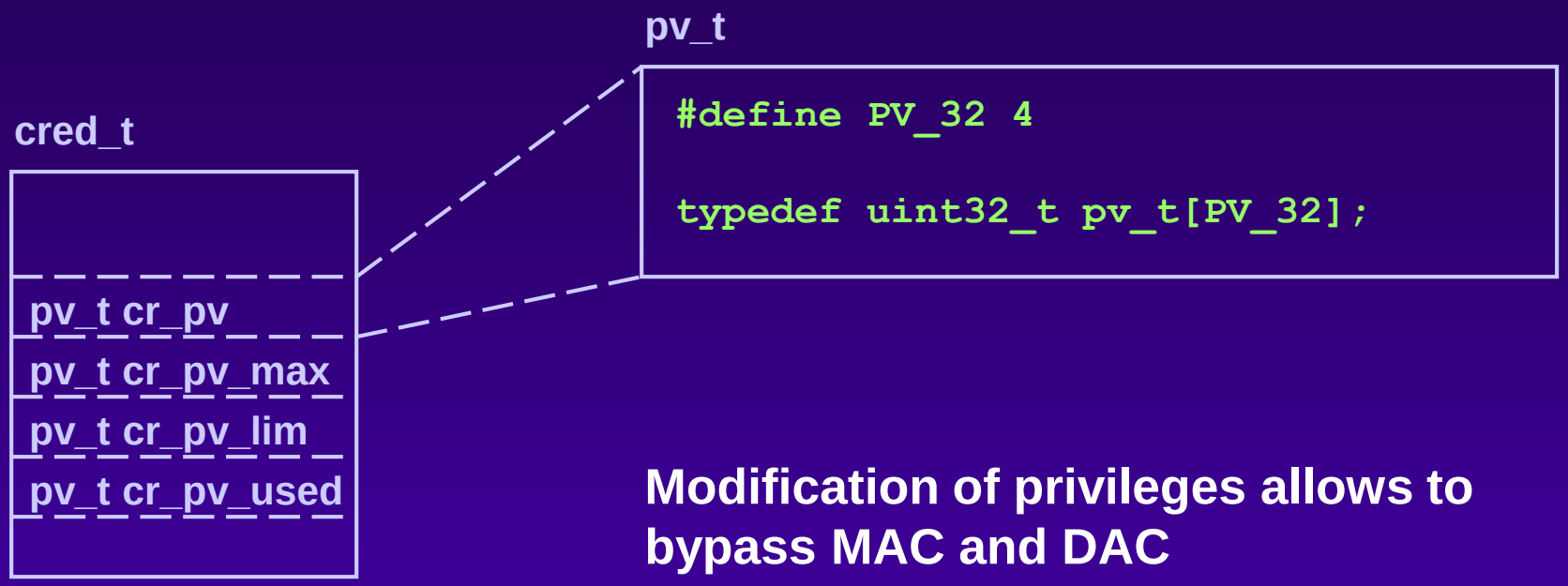
Modification: Pitbull sensitivity labels



To take control over MAC sl_class and sl_comp changes are needed

- changing classification requires setting one single integer value (sl_class)
- assignment to different compartments requires appropriate bits setting in sl_comp[ ] table
- correct ALL value must be selected according the system configuration

Modification: Pitbull privileges



Modification of privileges allows to bypass MAC and DAC

- in order to grant process all privileges, all bits in pv_t[ ] table must be set to 1,
- after exec() the Pitbull revokes all privileges for the security reasons,
- (sic!) the only exceptions are PV_ROOT* privileges, which are the most powerful and are inherited

SCO OpenServer setcontext() vulnerability

- Specific to x86 architecture and OS protection mechanisms provided by x86 family of processors
- A kernel level vulnerability, of which impact is very similar to the ldt bug
- Proper exploitation will result in a code execution at 0 protection level of a processor
- The setcontext() system call erroneously allows to set the CS code segment register of a given process to a user supplied value

The Conclusions

- The case study presented today is a good example of complexity of modern protection as well as attack techniques
- Existence of ldt kernel level vulnerability allowed to bypass security of Solaris system enhanced with Pitbull Foundation, which received various certifications (ITSec B1)
- This proves that there is no such thing like completely secured system
- The other thing is the range of practical impact that kernel level vulnerabilities might have

The Conclusions (LSD gains and losses)

+ Interesting experience, possibility of participation in great event,
+ An opportunity to meet interesting people,
+ Obviously the prize money (partly paid up to this day),
+ For Michael it was unforgettable bachelor party.

- Long sleepless night,
- 16 liters of Pepsi drunk by 4 persons within 24 hours may influence our health,
- An opportunity to meet interesting journalists.

Thank you for your attention

Last Stage of Delirium
Research Group

`http://lsd-pl.net`

contact@lsd-pl.net