The ORCHIDS Intrusion Prevention System

Jean Goubault-Larrecq, Julien Olivain

Laboratoire Spécification et Vérification
CNRS ENS Cachan
INRIA Futurs
SECSI

ARCS Meeting — SFI, Nov. 2, 2006

Outline

Some Types of Attacks

- Viruses, worms, trojans, buffer overflows, etc.
  (attacks on systems)
- Denial of service, IP/ARP spoofing, sniffing, etc.
  (network attacks)
- Attacks on html forms (perl, pgp), SQL insertion, IE/Word viruses [worms], etc.
  (attacks on applications)

Current Trends

Jean Goubault-Larrecq, Julien Olivain

- Systems and networks grow larger.

More and more difficult to ensure any level of security.

Current Trends

- Systems and networks grow larger.

  More and more difficult to ensure any level of security.

- Stakes are higher.

  On-line data-bases [banking, health, taxes, ...], e-commerce, etc.

Current Trends

- Systems and networks grow larger.

  More and more difficult to ensure any level of security.

- Stakes are higher.

  On-line data-bases [banking, health, taxes, ...],

  e-commerce, etc.

- Attacks are more and more sophisticated, automated, and distributed.

  Ready-to-use packages [ask Google]

  Attacks requiring several steps,

  . . . each of them being innocuous in isolation

Current Trends

Jean Goubault-Larrecq, Julien Olivain

- Systems and networks grow larger.

  More and more difficult to ensure any level of security.

- Stakes are higher.

  On-line data-bases [banking, health, taxes, ...],
  e-commerce, etc.

- Attacks are more and more sophisticated, automated, and distributed.

  Ready-to-use packages [ask Google]
  Attacks requiring several steps,
  ... each of them being innocuous in isolation

- New needs in intrusion detection

  Tracking user configurations
  Detecting internal fraud
  Smartcards

Julien Olivain

Hey, Jean! You're not just going to blabber along, right? Show them the **`ptrace`** attack for starters.

Julien Olivain

Hey, Jean! You're not just going to blabber along, right? Show them the **ptrace** attack for starters.

Er, that's what I meant to do… sure!

Me (confused)

Outline

The `ptrace` Attack [Purczyński01,03]

The ptrace Attack [Purczyński01,03]

Jean Goubault-Larrecq, Julien Olivain

Exploits a problem on rights of spawned processes, with the `ptrace` system call.
Effect: a local, user-to-root attack.

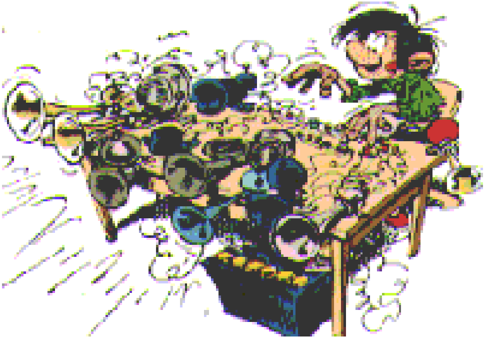
The `ptrace` Attack [Purczyński01,03]

Jean Goubault-Larrecq, Julien Olivain

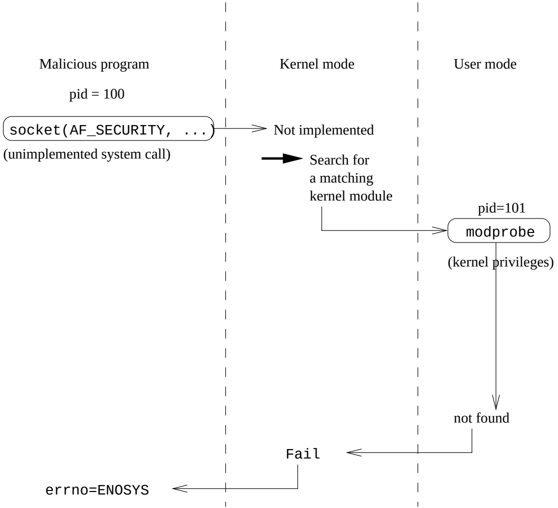
Exploits a problem on rights of spawned processes, with the `ptrace` system call.
Effect: a local, user-to-root attack.

- The `ptrace` call: used by all debuggers (benign in isolation!). Requires correlations.
- Subtle attack, based on a race condition in Linux 2.18 (Red Hat) kernels.

The Hacker's View



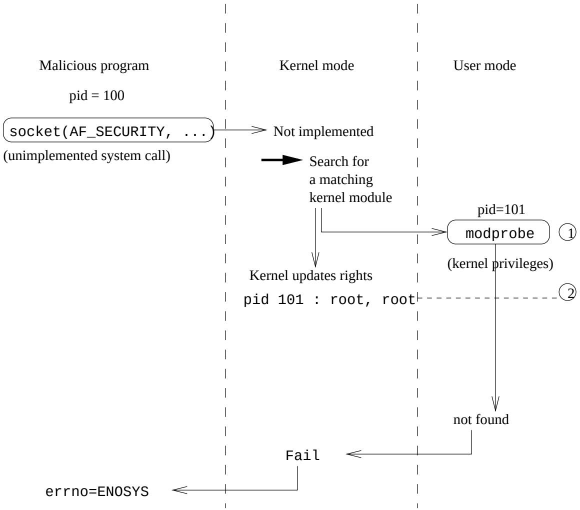
The ptrace Attack [Purczyński01,03]



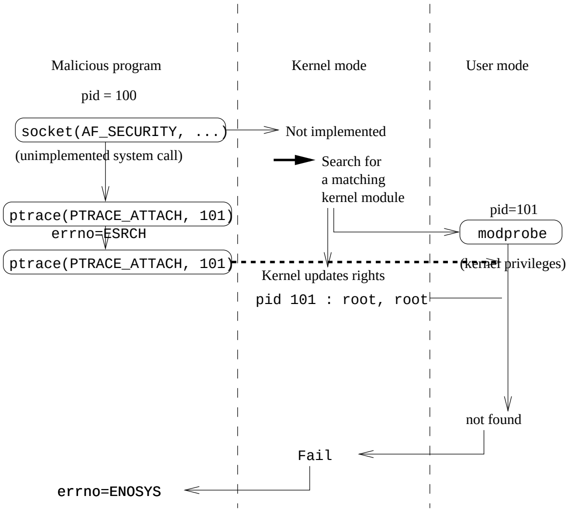
Jean Goubault-Larrecq, Julien Olivain

The ptrace Attack [Purczyński01,03]

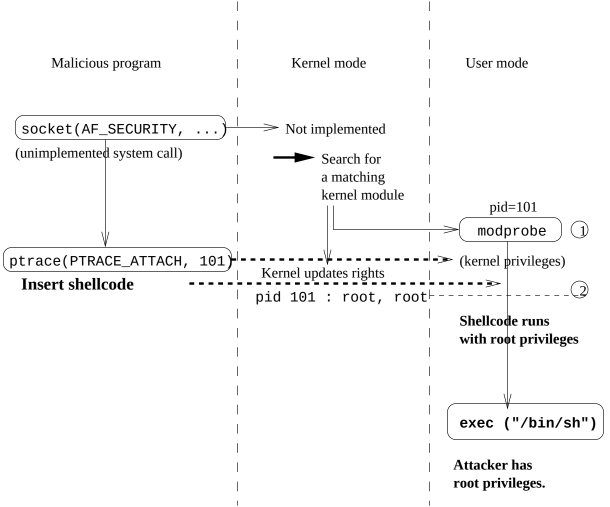
Jean Goubault-Larrecq, Julien Olivain



The ptrace Attack [Purczyński01,03]



The ptrace Attack [Purczyński01,03]



Outline

Jean Goubault-Larrecq, Julien Olivain

The Attack Signature

Jean Goubault-Larrecq, Julien Olivain

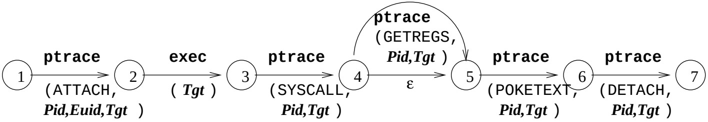
- We can count on the system logging important events.
  Here we count on the SNARE kernel module.
  We may also interface to the `syslog` facility.

The Attack Signature

- We can count on the system logging important events.

  Here we count on the SNARE kernel module.

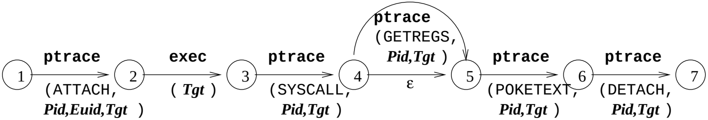
  We may also interface to the `syslog` facility.
- ORCHIDS will now try to find patterns among these logged events:



The Attack Signature

Jean Goubault-Larrecq, Julien Olivain

- We can count on the system logging important events.
  Here we count on the SNARE kernel module.
  We may also interface to the `syslog` facility.
- ORCHIDS will now try to find patterns among these logged events:



- Note that just detecting calls `ptrace` is not enough: this is used in everyday debugging activities, and is not indicative of an attack by itself.

Demo

Julien Olivain

Now let's see Orchids in action.

Demo — Reacting to an Intrusion

Julien Olivain

Jean, did you show them that Orchids *killed* the offending user's account?
Did you explain them why?

Demo — Reacting to an Intrusion

Julien Olivain

Jean, did you show them that Orchids *killed* the offending user's account?
Did you explain them why?

Sure, Julien:

- The attacker may have left a backdoor in the system, allowing him to reenter at will.
  We should prevent him from using it later.
- Also, Orchids produces reports!
  Here, tracks attacker's achievements.

Demo — Escaping Masking Attacks

Jean Goubault-Larrecq, Julien Olivain

Demo — Escaping Masking Attacks

Julien Olivain

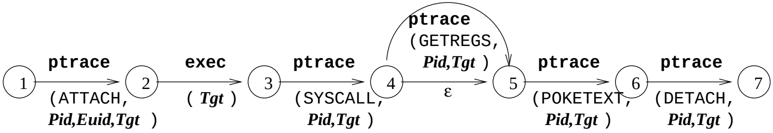
Jean, have you shown them that Orchids was not fooled by *masking* attacks?

Oh yes. The point is that the attacker may attempt to drown the intruder detection system under many similar events.

Goal: attempt to escape detection.

Let's see how Orchids fares under pressure : let's generate zillions of benign calls to **`ptrace`**.

How Does ORCHIDS Detect It?



Imagine the following flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

(What to come, in a nutshell: the ORCHIDS engine will basically simulate a form of alternating automata, with additional optimizations gotten by abstract interpretation of the formulae.)

Detecting the Attack

Jean Goubault-Larrecq, Julien Olivain

Flow of events:

`open` ("/etc/passwd", "r", pid=58, euid=500)	`ptrace` (SYSCALL, pid=100, 101)
`ptrace` (ATTACH, pid=57, euid=500, 58)	`ptrace` (GETREGS, pid=100, 101)
`ptrace` (ATTACH, pid=100, euid=500, 101)	`ptrace` (POKETEXT, pid=100, 101)
`exec` (prog="modprobe", pid=101)	`ptrace` (POKETEXT, pid=100, 101)
`ptrace` (ATTACH, pid=100, euid=500, 101)	`ptrace` (POKETEXT, pid=100, 101)
`exit` (pid=58)	`ptrace` (DETACH, pid=100, 101)

Initially, ORCHIDS has no active thread.

Detecting the Attack

Jean Goubault-Larrecq, Julien Olivain

Flow of events:

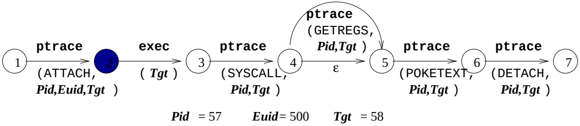
open ("/etc/passwd", "r", pid=58, euid=500)	ptrace (SYSCALL, pid=100, 101)
ptrace (ATTACH, pid=57, euid=500, 58)	ptrace (GETREGS, pid=100, 101)
ptrace (ATTACH, pid=100, euid=500, 101)	ptrace (POKETEXT, pid=100, 101)
exec (prog="modprobe", pid=101)	ptrace (POKETEXT, pid=100, 101)
ptrace (ATTACH, pid=100, euid=500, 101)	ptrace (POKETEXT, pid=100, 101)
exit (pid=58)	ptrace (DETACH, pid=100, 101)

The signature contains no open event: skip.

Detecting the Attack

Flow of events:

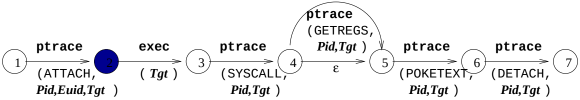
open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)



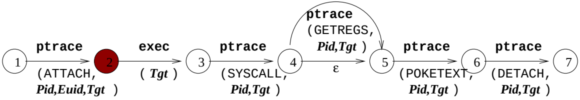
Detecting the Attack

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)



Pid = 57 *Euid* = 500 *Tgt* = 58



Pid = 100 *Euid* = 500 *Tgt* = 101

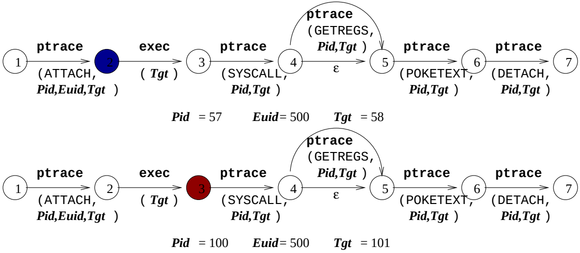
Spawn thread:
avoid masking attacks.

Detecting the Attack

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)



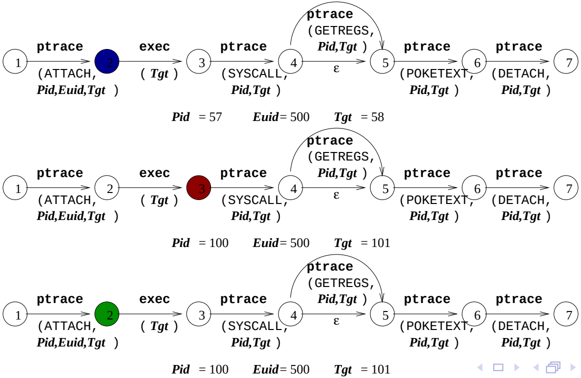
No need to spawn thread: would violate shortest runs.

Detecting the Attack

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)



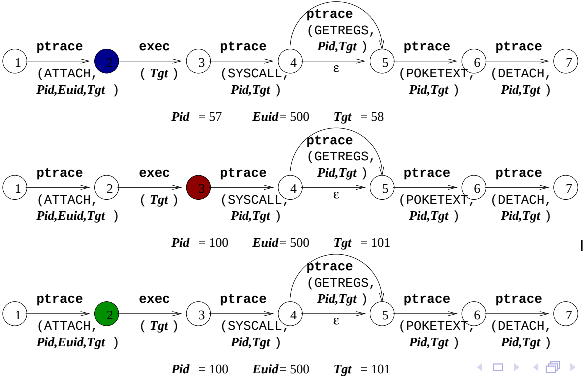
Detecting the Attack

Jean Goubault-Larrecq, Julien Olivain

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)



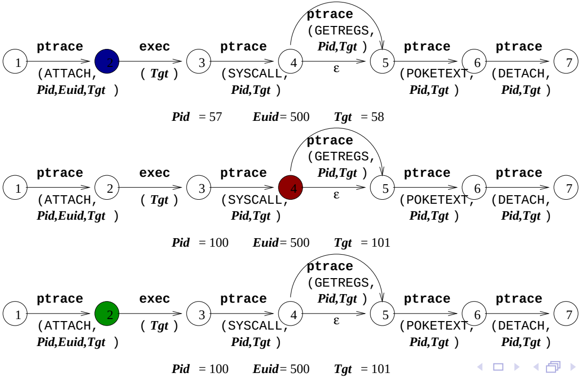
Irrelevant event exit

Detecting the Attack

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

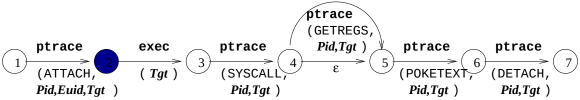


Detecting the Attack

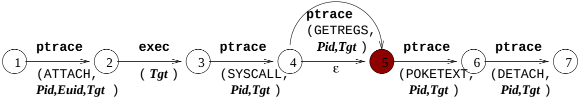
Jean Goubault-Larrecq, Julien Olivain

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)
ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)

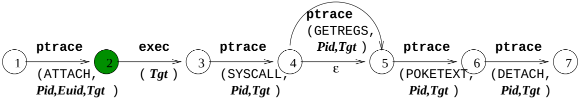


Pid = 57 *Euid* = 500 *Tgt* = 58



Pid = 100 *Euid* = 500 *Tgt* = 101

Keep run ATTACH-exec-SYSCALL-GETREGS for reporting.



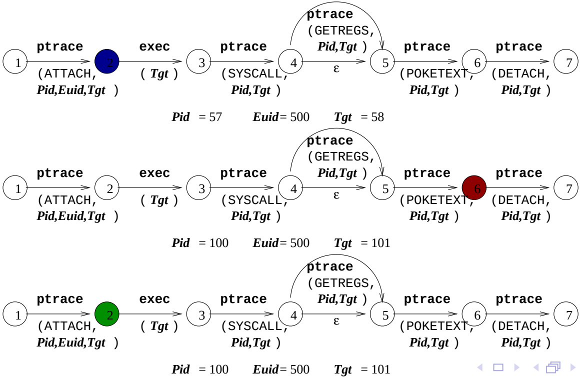
Pid = 100 *Euid* = 500 *Tgt* = 101

Detecting the Attack

Jean Goubault-Larrecq, Julien Olivain

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)	ptrace (SYSCALL, pid=100, 101)
ptrace (ATTACH, pid=57, euid=500, 58)	ptrace (GETREGS, pid=100, 101)
ptrace (ATTACH, pid=100, euid=500, 101)	ptrace (POKETEXT, pid=100, 101)
exec (prog="modprobe", pid=101)	ptrace (POKETEXT, pid=100, 101)
ptrace (ATTACH, pid=100, euid=500, 101)	ptrace (POKETEXT, pid=100, 101)
exit (pid=58)	ptrace (DETACH, pid=100, 101)



Do not spawn thread (shortest runs again) (prefer A– over -A-, –A on trace AAA). Formally: keep lexicographically smallest sequences of event numbers.

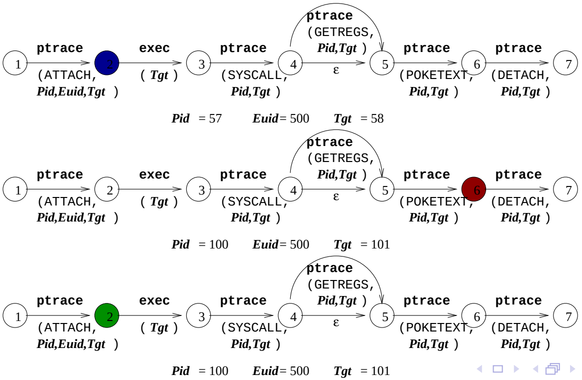
Jean Goubault-Larrecq, Julien Olivain

Detecting the Attack

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)



Do not spawn thread (shortest runs again) (prefer A– over -A-, –A on trace AAA). Formally: keep lexicographically smallest sequences of event numbers.

Detecting the Attack

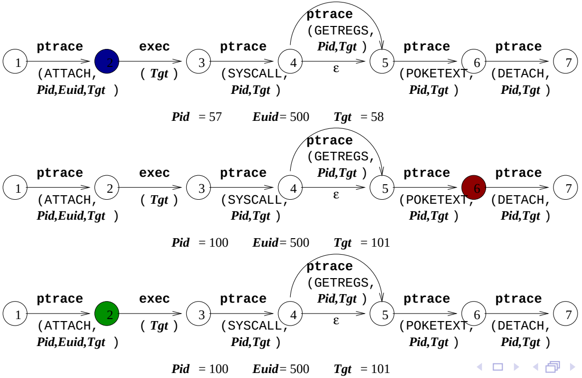
Jean Goubault-Larrecq, Julien Olivain

Issues
Example Attack
Running ORCHIDS
Demo
Under the Hood
Architecture
Beyond
Way Beyond
Conclusion
Misc

Flow of events:

open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)



Do not spawn thread (shortest runs again) (prefer A– over -A-, –A on trace AAA). Formally: keep lexicographically smallest sequences of event numbers.

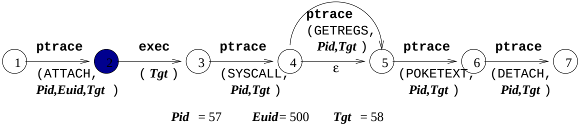
Detecting the Attack

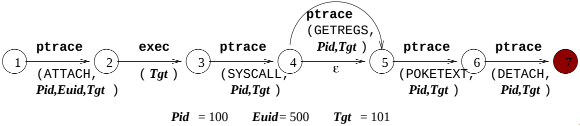
Jean Goubault-Larrecq, Julien Olivain

Flow of events:

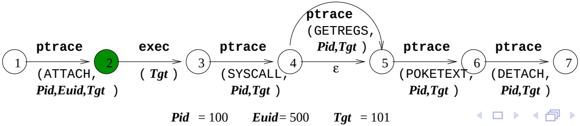
open ("/etc/passwd", "r", pid=58, euid=500)
ptrace (ATTACH, pid=57, euid=500, 58)
ptrace (ATTACH, pid=100, euid=500, 101)
exec (prog="modprobe", pid=101)
ptrace (ATTACH, pid=100, euid=500, 101)
exit (pid=58)

ptrace (SYSCALL, pid=100, 101)
ptrace (GETREGS, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (POKETEXT, pid=100, 101)
ptrace (DETACH, pid=100, 101)





Alert.

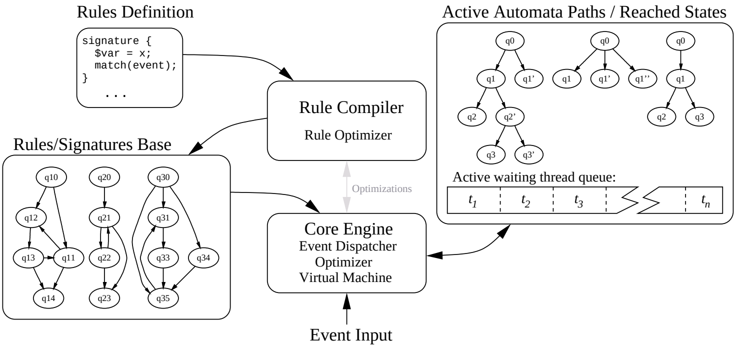


Jean Goubault-Larrecq, Julien Olivain

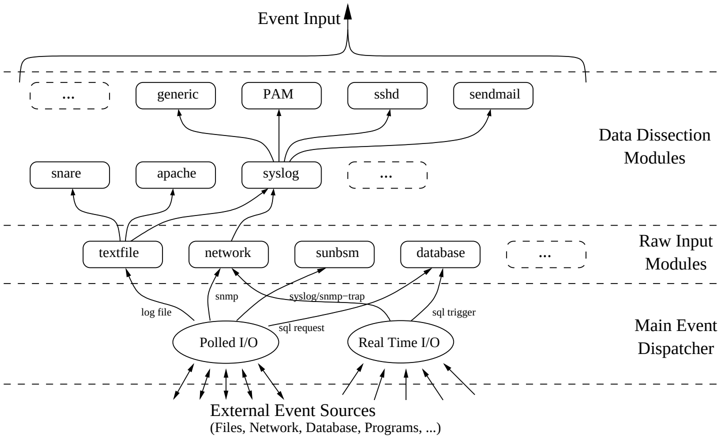
Outline

A Bird's Eye View

- Efficient implementation, through compilation to bytecode trees.



A Modular Architecture



Input Sources

Jean Goubault-Larrecq, Julien Olivain

Multi-sensor:

- System sensors:
  - SNARE, `syslog` (Unix);
  - MS EVT (Windows);
- Network and equipment sensors:
  - Cisco logs;
  - SNMP sensors;
  - Linux NetFilter.
  - ...
- Meta-sensors: e.g., SNORT used as a sensor.
- Filters: e.g., NetEntropy entropy tester.
- ...

Input Sources

Jean Goubault-Larrecq, Julien Olivain

Multi-sensor:

- System sensors:
  - SNARE, `syslog` (Unix);
  - MS EVT (Windows);
- Network and equipment sensors:
  - Cisco logs;
  - SNMP sensors;
  - Linux NetFilter.
  - ...
- Meta-sensors: e.g., SNORT used as a sensor.
- Filters: e.g., NetEntropy entropy tester.
- ...

Multi-port: reads from UDP connections, SNMP connections, log files.

Outline

ORCHIDS

Jean Goubault-Larrecq, Julien Olivain

Cuts

Cuts (à la Prolog): generalize shortest runs.
Allow one to specify removing threads thought to be irrelevant.

Jean Goubault-Larrecq, Julien Olivain

Cuts

Jean Goubault-Larrecq, Julien Olivain

Cuts (à la Prolog): generalize shortest runs.
Allow one to specify removing threads thought to be irrelevant.

Green cuts: Preserve the *no-masking* property:

Each attack (family of runs) $\Rightarrow$ at least one alert is reported.

Shortest runs are an example, with an additional *optimality* property: exactly one alert is reported.

Cuts

Jean Goubault-Larrecq, Julien Olivain

Cuts (à la Prolog): generalize shortest runs.
Allow one to specify removing threads thought to be irrelevant.

Green cuts: Preserve the *no-masking* property:

Each attack (family of runs) $\Rightarrow$ *at least one alert is reported.*

Shortest runs are an example, with an additional *optimality* property: exactly one alert is reported.

Red cuts: Forget attacks.

- Avoid congestion, avert attacks on the IDS.
- Simulate monitors. (case where no thread is spawned dynamically.)
- Intermediate reports. (emit them, kill threads having emitted none, then proceed.)
- Implement the `without` operator (check that no event satisfying F occurs while waiting for G).

(e.g., kill threads when monitored process exits.)

Not Just One Attack, But Families

Julien Olivain

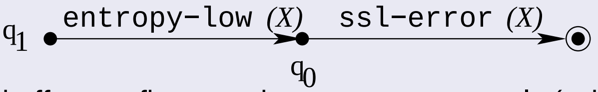
Jean, I'm afraid you did not insist on the fact that you could catch whole *families* of attacks by just one ORCHIDS signature rule. Have you demonstrated the **do_brk** attack, by the way?

Detect Families of Attacks

Jean Goubault-Larrecq, Julien Olivain

Signatures can be made to match not just one attack, rather whole families.

The Entropy Checker



detects all buffer overflow attacks on crypto protocols (ssh1, ssh2, https, ldaps, ...).

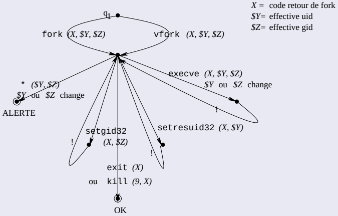
- By the way, this is a network attack. ORCHIDS is not limited to system attacks.
- By the way, it uses the Net-Entropy sensor to detect entropy anomalies in input flow.

Detect Families of Attacks

Jean Goubault-Larrecq, Julien Olivain

Signatures can be made to match not just one attack, rather whole families.

The Pid Tracker



detects all attacks in the style of `do_brk`, `mmap`, `munmap`, `mremap`, etc. [MortonStarzetz'03].

- Principle: detect that some user gained root privileges without using the authorized `set*id` mechanism, whichever way he actually managed to do so.
- Technically, a form of dynamically-spawned monitors.

The Attacks on do_brk, mmap, munmap, . . . [MortonStarzetz03]

Jean Goubault-Larrecq, Julien Olivain

One of the most serious attacks ever actually used.

- Crackers used it to bog down the Savannah (GNU) and Debian servers in 2004. Downtime: several weeks.

The Attacks on `do_brk`, `mmap`, `munmap`, … [MortonStarzetz03]

Jean Goubault-Larrecq, Julien Olivain

One of the most serious attacks ever actually used.

- Crackers used it to bog down the Savannah (GNU) and Debian servers in 2004. Downtime: several weeks.
- Vicious attacks: give rise to no event at all.

  …except a flurry of calls to `do_brk` (but this is what `malloc` calls!)

  …except a flurry of `SIGSEGV` signals (not logged by SNARE!)
- Principle: rewrite the information the kernel keeps on our (user) process by mapping the kernel into the address space of the process (!).

Outline

Information Correlation

Jean Goubault-Larrecq, Julien Olivain

Use an embedded Prolog interpreter. Applications include:

Keep set of attacks that have already succeeded.

- Maintain Black Lists.
- Realize that user *A* has succeeded in the past in mounting an attack on machine *M* giving him a user account,
  allowing him to try and launch a second attack on *M* in the hope of gaining root privileges.

Information Correlation

Jean Goubault-Larrecq, Julien Olivain

Use an embedded Prolog interpreter. Applications include:

Reason about network topology.

- realize that `127.0.0.1` and `localhost` are the same machine (as in the M2D2 model [MMDD03]);
- realize that two machines *A* and *B* are neighbors and may have cooperated in mounting an attack.

Outline

Jean Goubault-Larrecq, Julien Olivain

Conclusion

ORCHIDS, an efficient on-line, real-time intrusion prevention system.

- Handles real, recent, sophisticated attacks;
- Produces detailed reports, runs countermeasures;

  i.e., emergency measures until the sys. admin. reacts.
- Multi-sensor, multi-port, multi-event. . . multi-whatever.

Conclusion

ORCHIDS, an efficient on-line, real-time intrusion prevention system.

- Handles real, recent, sophisticated attacks;
- Produces detailed reports, runs countermeasures;

  i.e., emergency measures until the sys. admin. reacts.
- Multi-sensor, multi-port, multi-event. . . multi-whatever.

Care to join? Contact:

Julien Olivain	Jean Goubault-Larrecq
`olivain@lsv.ens-cachan.fr`	`goubault@lsv.ens-cachan.fr`

Outline

Jean Goubault-Larrecq, Julien Olivain

The `mod_ssl` Attack [McDonald03]

Jean Goubault-Larrecq, Julien Olivain

Exploits a buffer overflow in the OpenSSL code for Apache, implementing SSL v.2.
Effect: remote exploit, obtaining a remote shell.

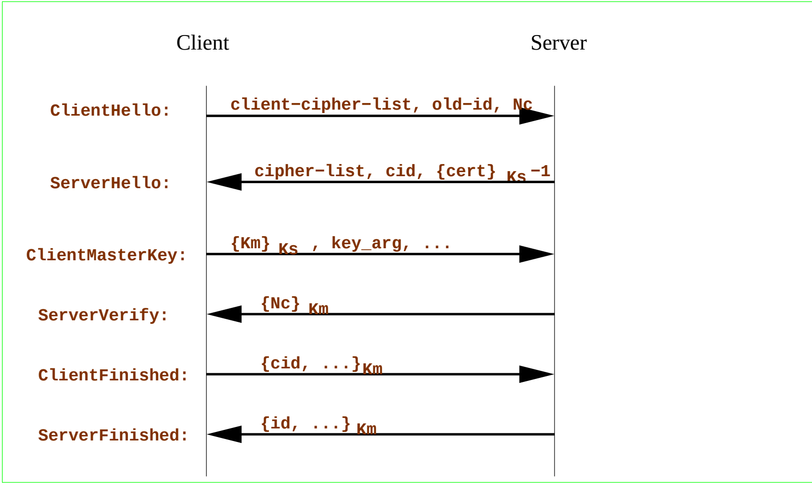
The mod_ssl Attack [McDonald03]

Jean Goubault-Larrecq, Julien Olivain

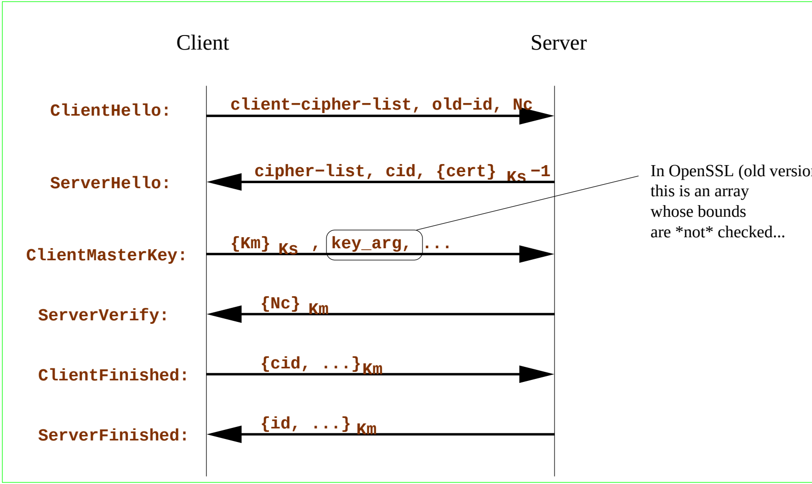
Exploits a buffer overflow in the OpenSSL code for Apache, implementing SSL v.2.
Effect: remote exploit, obtaining a remote shell.

- Extremely complex attack.
- Exploits several Apache threads to get vulnerability information.
- Transmits vulnerability information through encrypted SSL channel.

The SSL v2 Handshake Protocol



An (Important) Detail of Implementation in OpenSSL



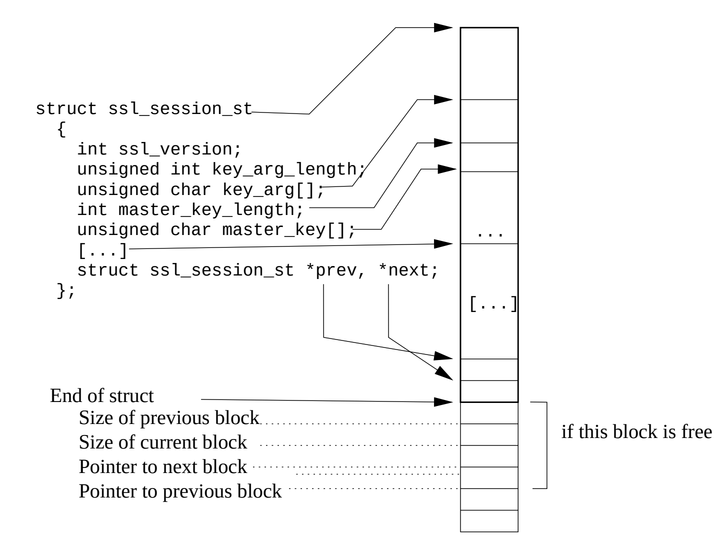
An Attack on OpenSSL

Jean Goubault-Larrecq, Julien Olivain

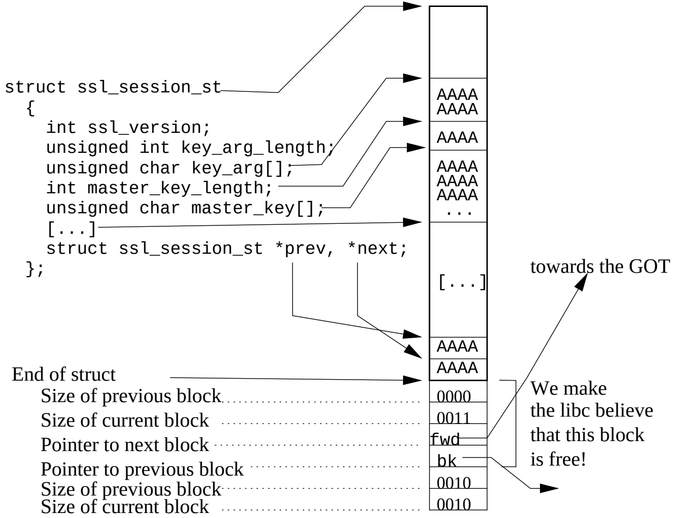
When the server receives `ClientMasterKey`, it copies it into:

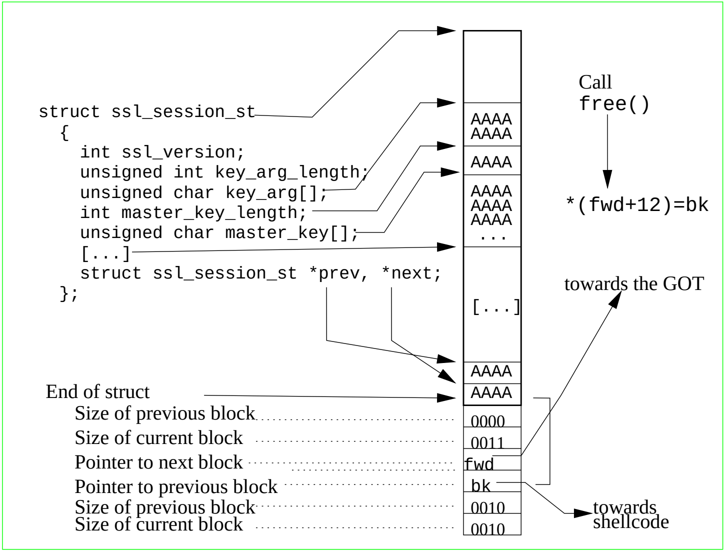
```
typedef struct ssl_session_st
    {
    int ssl_version;
    unsigned int key_arg_length;
    unsigned char key_arg[SSL_MAX_KEY_ARG_LENGTH];
    int master_key_length;
    unsigned char master_key[SSL_MAX_MASTER_KEY_LENG
    [...]
    struct ssl_session_st *prev, *next;
    } SSL_SESSION;
```

Vulnerable: let's stuff it!


struct ssl_session_st
{
int ssl_version;
unsigned int key_arg_length;
unsigned char key_arg[];
int master_key_length;
unsigned char master_key[];
[...]
struct ssl_session_st *prev, *next;
};
...
[...]
End of struct
Size of previous block
Size of current block
Pointer to next block
Pointer to previous block
if this block is free

Jean Goubault-Larrecq, Julien Olivain



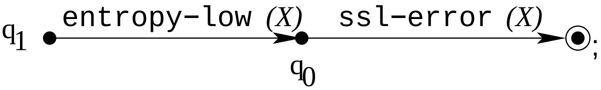

struct ssl_session_st
{
int ssl_version;
unsigned int key_arg_length;
unsigned char key_arg[];
int master_key_length;
unsigned char master_key[];
[...]
struct ssl_session_st *prev, *next;
};
AAAA
AAAA
AAAA
AAAA
AAAA
AAAA
...
[...]
AAAA
AAAA
End of struct
Size of previous block
Size of current block
Pointer to next block
Pointer to previous block
Size of previous block
Size of current block
0000
0011
fwd
bk
0010
0010
Call
free()
*(fwd+12)=bk
towards the GOT
towards
shellcode

The rest of the attack, in short

Jean Goubault-Larrecq, Julien Olivain

- To retrieve the address of the shellcode, have the server retransmit all needed information by writing into the `session_id` field: we get the information, encrypted, in the `ServerFinished` message.
- We now know at which addresses the server works.
- Now replay a similar attack in a second SSL session to really execute the shellcode.
- The server now serves a connection to an `apache` or `nobody` shell through HTTP. (Now play a user-to-root attack. . .)

Describing the attack

- 

- Note: use an entropy input module (to be published);
- Detects any attack on SSL where plain texts are served instead where we expect ciphertexts.