The ORCHIDS Intrusion Detection Tool[⋆]

Julien Olivain Jean Goubault-Larrecq

LSV/CNRS UMR 8643 & INRIA Futurs projet SECSI & ENS Cachan
61 avenue du président-Wilson, F-94235 Cachan Cedex
olivain@lsv.ens-cachan.fr
Phone: +33-1 47 40 75 50 Fax: +33-1 47 40 24 64

Abstract. ORCHIDS is an intrusion detection tool based on techniques for fast, on-line model-checking. Temporal formulae are taken from a temporal logic tailored to the description of intrusion signatures. They are checked against merged network and system event flows, which together form a linear Kripke structure.

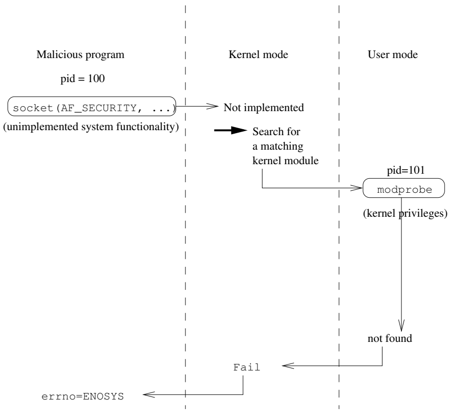
1 Introduction: Misuse Detection as Model-Checking

ORCHIDS is a new intrusion detection tool, capable of analyzing and correlating events over time, in real time. Its purpose is to detect, report, and take countermeasures against intruders. The core of the engine is based on the language and algorithm in the second part of the paper by Muriel Roger and Jean Goubault-Larrecq [5]. The precise algorithm is described in two reports [2, 1].

The point, explained in [5], is that intrusion detection, and specifically *misuse detection*, whereby bad behavior (so-called *attacks*) is specified in some language and alerts are notified when bad behavior is detected, is essentially a *model-checking* task. The Kripke model to be analyzed is an *event flow* (collected from various logs, and other system or network sources), and complex attack *signatures* are described in some application-specific temporal logic.

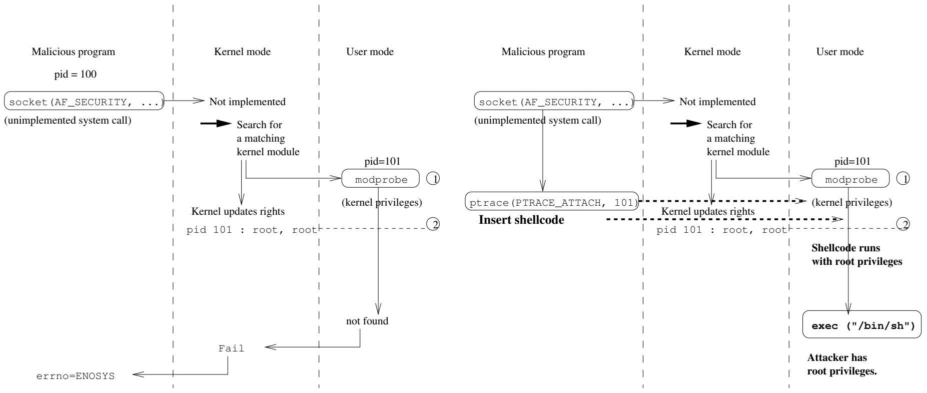
Let us give an example of a modern attack [4]. Far from being a gedankenexperiment, this really works in practice and has already been used to penetrate some systems. We also insist that, as systems get more and more secure, we are faced with more and more complex attacks, and [4] is just one representative.

The schema on the right displays what a modular kernel (e.g., Linux) does when a user program (here with pid 100) calls an unimplemented functionality.



[⋆] Partially supported by the RNTL Project DICO, the ACI jeunes chercheurs "Sécurité informatique, protocoles cryptographiques et détection d'intrusions" and the ACI cryptologie "Psi-Robuste".

The kernel will search for a kernel module that implements this functionality, calling the `modprobe` utility to search and install the desired module. If `modprobe` does not find any matching module, an error code is reported to the user program.



(a) A race condition in the kernel

(b) The final exploit

Fig. 1. The `ptrace` Linux attack

While this is how this is meant to work, some versions of Linux suffer from a race condition (Figure 1(a)): while `modprobe` has all kernel privileges, the kernel updates the owner tables to make `modprobe` root-owned instead of user-owned while `modprobe` has already started running. So there is a small amount of time where the malicious program has complete control over the kernel process `modprobe`: between timepoints ① and ②. The malicious program takes this opportunity to attach the `modprobe` process through the standard Unix debugging API function `ptrace`, inserting a *shellcode* (malicious code) inside `modprobe`'s code. When `modprobe` resumes execution, it will execute any code chosen by the intruder, with full root privileges. This is summed up in Figure 1(b).

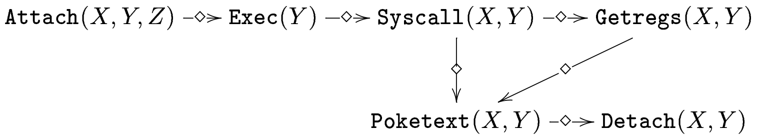
2 Challenges in On-Line, Real-Time Model-Checking

Intrusion detection requires specific logics to describe attack signatures, and specific model-checking algorithms: see [5].

Compared to standard misuse detection tools, a temporal logic allows one to describe behavior involving several events over time: standard misuse detection tools (e.g., anti-virus software or simple network intrusion detection systems) match a library of patterns against single events, and emit an alert once single so-called *dangerous* events occur. More and more attacks nowadays involving complex, correlated sequences of events, which are usually individually benign. In the `ptrace` attack, *no* individual event (calling an unimplemented system call, calling ptrace, etc.) is dangerous per se.

The second logic of [5], which is the basis for the signature language of ORCHIDS, allows one to write temporal formulas of the typical form $F_1 \wedge \Diamond(F_2 \wedge \Diamond(F_3 \ldots) \vee F'_2 \wedge \Diamond(F'_3 \ldots))$ in particular, where $\Diamond$ is the strict "there exists in the future" operator. In general, more complex formulae can be written, using operators resembling Wolper's ETL [6]—except going through a transition denotes either no time-passing at all (ϵ-transitions), or $\Diamond$ (not $\bigcirc$ as in ETL). Such formulae are described as finite automata; we just give a signature for the ptrace exploit as an illustration.

A formula matching the ptrace exploit is the following automaton, described in slightly idealized form:



(1)

where X, Y, Z are existentially quantified first-order variables meant to match the attacker's pid, the target's pid (i.e., modprobe's pid), and the attacker's effective uid respectively; where $\texttt{Attach}(X, Y, Z)$ abbreviates a formula (not shown) matching any single event displaying a call to ptrace by process X owned by Z, on process Y, with the ATTACH command, $\texttt{Exec}(Y)$ matches single events where /sbin/modprobe is execed with pid Y, and the remaining formulas match single events where process X issues a call to ptrace on target Y, with respective commands SYSCALL, GETREGS, POKETEXT (used to insert the shellcode), and DETACH.

Compared to other more standard uses of model-checking in verification, the logic of ORCHIDS is constrained to only specify *eventuality* properties. This is because the model-checker needs to to work *on-line*, that is, by always working on some finite (and expanding over time) prefix of an infinite sequence of events. Accordingly, formulae such as (1) are built using $\wedge$, $\vee$, and $\Diamond$ only, and no negation.

Compared to standard model-checking algorithms, e.g., based on Büchi automata for LTL, the model-checker is not allowed to make multiple passes over the sequence of events (e.g., we cannot rebuild a product automaton each time a new event is added); in general, intrusion detection tasks are submitted to very stringent efficiency requirements, both in time and in space.

Second, the logic of ORCHIDS includes some first-order features. As witnessed by the use of variables X, Y, Z in (1), this logic can be seen as an existential fragment of a first-order temporal logic.

Finally, such a model-checker cannot just report the *existence* of matches, but must enumerate all matches among a given representative subset, with the corresponding values of the existential variables, build an alert for each match and possibly trigger countermeasures. This is the *raison d'être* behind the $\texttt{Getregs}(X, Y)$ formula in (1); if we only wanted a yes/no answer, this would just be redundant, and could be erased from the automaton; here, this is used to be able to report whether the attacker issued at least one call to ptrace(PTRACE_GETREGS) or not during the attack.

The model-checking task for the logic of ORCHIDS is NP-complete, but can be done using an efficient, on-line and real-time algorithm [5, 2, 1]. Moreover, this algorithm is *optimal* [5, Theorem 4.11] in the following sense: for every attack signature (formula F), if at least one attack (sequence of possibly non-contiguous events) is started at event e_0 that matches F, then exactly one attack is reported amongst these, the one with the so-called *shortest run*. In practice, the latter is always the most meaningful attack among all those that match.

3 Implementation

The ORCHIDS engine is implemented in C. At the core of ORCHIDS lies a fast virtual machine for a massively-forking virtual parallel machine, and a bytecode compiler from formulae (such as (1)) to this virtual machine. ORCHIDS uses a hierarchy of input modules to subscribe to, and to parse incoming events, classified by input source and/or event format. A main event dispatcher reads from polled and real-time I/O, reads sequences of events in `syslog` format, `snare`, `sunbsm`, `apache` and other various formats, coming from log files or directly through dedicated network connections, and feeds the relevant events to the core engine. ORCHIDS is able to do both system-level and network-based intrusion detection, simultaneously.

4 Conclusion

The `ptrace` attack above is one of the typical attacks that ORCHIDS can detect. Experiments are going on at LSV to test ORCHIDS on actual network traffic and system event flows. For more information, see the Web page [3].

References

1. J. Goubault-Larrecq. Un algorithme pour l'analyse de logs. Research Report LSV-02-18, Lab. Specification and Verification, ENS de Cachan, Cachan, France, Nov. 2002. 33 pages.
2. J. Goubault-Larrecq, J.-P. Pouzol, S. Demri, L. Mé, and P. Carle. Langages de détection d'attaques par signatures. Sous-projet 3, livrable 1 du projet RNTL DICO. Version 1, June 2002. 30 pages.
3. J. Olivain. ORCHIDS—real-time event analysis and temporal correlation for intrusion detection in information systems. `http://www.lsv.ens-cachan.fr/orchids/`, 2004.
4. W. Purczyński. Linux kernel privileged process hijacking vulnerability. `http://www.securityfocus.com/bid/7112`, Mar. 2003. BugTraq Id 7112.
5. M. Roger and J. Goubault-Larrecq. Log auditing through model checking. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01), Cape Breton, Nova Scotia, Canada, June 2001*, pages 220–236. IEEE Comp. Soc. Press, 2001.
6. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983.

A Appendix

A.1 Tool demonstration

The demonstration will emphasize the most important aspects of the intrusion detection tool ORCHIDS. In addition to the points stressed in the tool paper itself, and in particular the challenges that have to be met for an on-line, real-time model-checker in such specific environments, we want to show:

The large range of data sources and types: ORCHIDS can extract data and events from multiple distributed sources (kernel system calls, network firewall actions, web server, mail server, etc.) This wide variety of events enables ORCHIDS to correlate many events over time *and* over different event sources, a particularly welcome asset in a modern intrusion detection system. The demonstration shows the modularity of ORCHIDS; it currently recognizes 400 data fields from about 50 data sources.

On-line analysis: ORCHIDS continuously receives events from sources, as soon as possible after their creation date (the delay depends of the communication method for reporting events, either local or remotely, through `udp`). The analysis has to be done *in one pass*, *incrementally*, over the event flow. The event flow, as seen by the analyzer, will be shown during the demonstration.

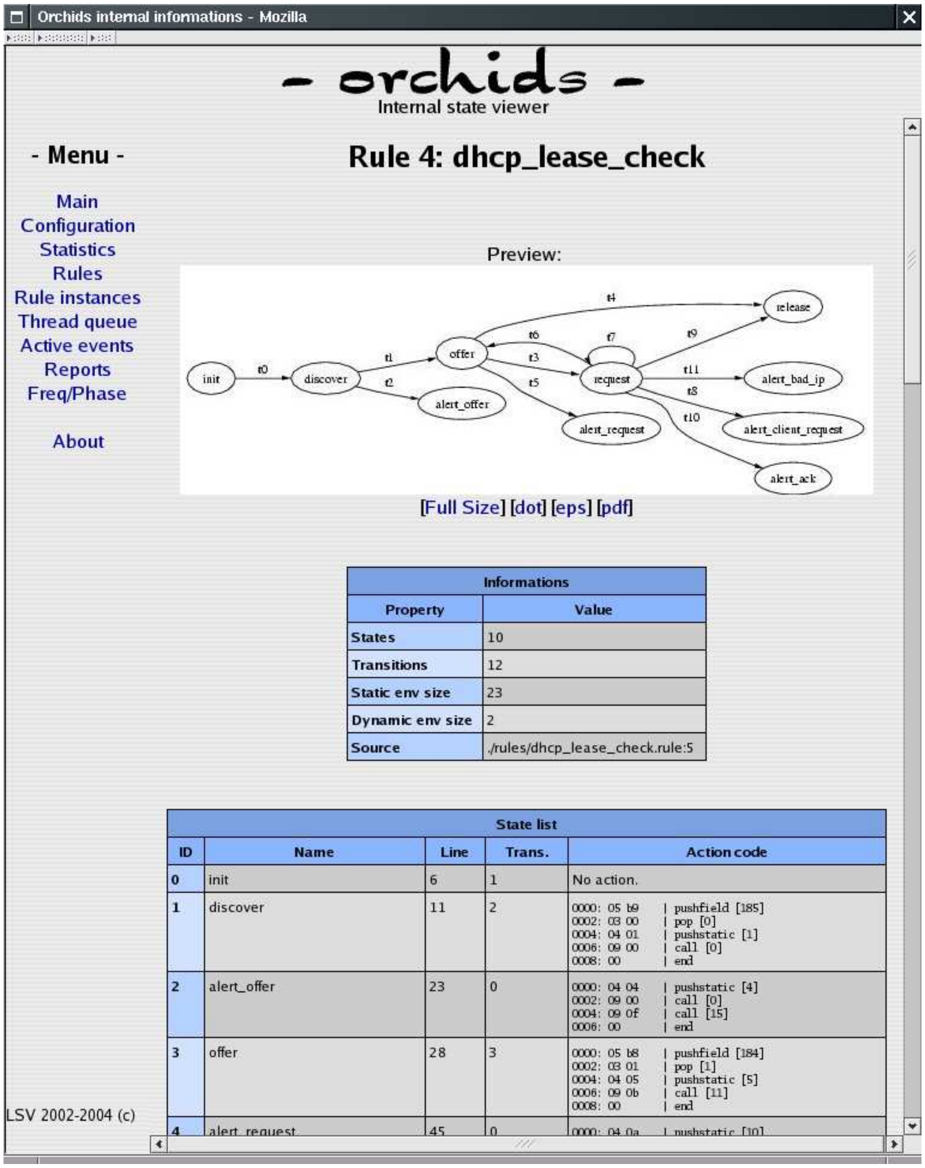
Real-time detection and coutermeasure: ORCHIDS collects data, computes information, and executes actions at any given instant of the detection process. ORCHIDS is organized so as to react as quickly as possible to an attack, and must therefore announce that an attack has been detected as soon as the last characteristic event of the attack has been received. Additionally, ORCHIDS is able to generate intermediate reports, when only part of a temporal formula has been matched, i.e., when it is only *likely* that a given attack is currently being perpetrated. If the temporal formula eventually matches in whole, ORCHIDS will eventually confirm that the attack occurred, through a complementary report.
Moreover, we shall demonstrate that ORCHIDS is able to take countermeasures against attackers by killing compromised processes, closing suspect user accounts, or blocking network connections by inserting new rules into a firewall.

These features will be demonstrated by running ORCHIDS itself, and by inspecting the formulae, the currently monitored instances of these formulae (the *threads*), the current event flow, through ORCHIDS' administration interface (see Figure 2).

It is to be noted, too, that the attacks we shall demonstrate are no toy examples. They are recent, sophisticated attacks, which have already been used in practice (by hackers), and whose detection is particularly challenging.

The demonstration is done on a single laptop. Demonstrating a distributed intrusion detection system working on just one computer entails some specific difficulties or deviations from real behavior:

- The analyzer runs on the audited computer; so it will analyze its own event flow, as well as its sensors'. Some precautions have to be taken to avoid the analyzer to loop, analyzing itself and therefore generating new events that have to be monitored, . . . , recursively.



Rule 4: dhcp_lease_check

Preview:

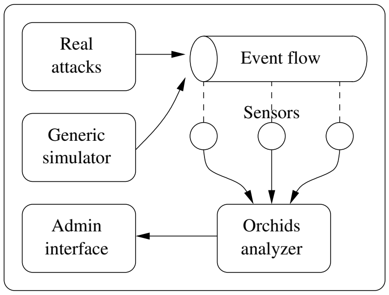
Informations	
Property	**Value**
States	10
Transitions	12
Static env size	23
Dynamic env size	2
Source	./rules/dhcp_lease_check.rule:5

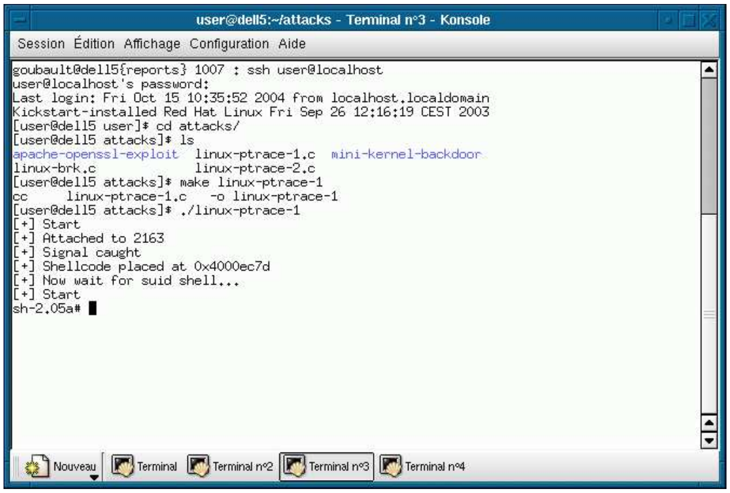
State list				
ID	**Name**	**Line**	**Trans.**	**Action code**
0	init	6	1	No action.
1	discover	11	2	0000: 05 b9 | pushfield [185] 0002: 03 00 | pop [0] 0004: 04 01 | pushstatic [1] 0006: 09 00 | call [0] 0008: 00 | end
2	alert_offer	23	0	0000: 04 04 | pushstatic [4] 0002: 09 00 | call [0] 0004: 09 0f | call [15] 0006: 00 | end
3	offer	28	3	0000: 05 b8 | pushfield [184] 0002: 03 01 | pop [1] 0004: 04 05 | pushstatic [5] 0006: 09 0b | call [11] 0008: 00 | end
4	alert_request	45	0	0000: 04 0a | pushstatic [10]

Fig. 2. The ORCHIDS administration interface

- The real attacks we shall demonstrate are launched against the operating system kernel, which can then end up in an unstable state. We have been careful to craft our attacks so as to be *perfect* (from the attacker point of view), i.e., to steal the target resource without crashing the whole system. In actual deployments (and in our testbed at LSV), ORCHIDS would naturally be running on a remote machine with only in connections to receive event flows, so that crashing an audited system would not crash ORCHIDS.
- The activity of a single host is much less intense than that of an entire network. To simulate the event flow in a realistic network environment, we shall use the *EvtGen* tool (which we designed specifically to this end) to generate fake events, too, simulating normal behaviour with randomly interspersed errors or anomalies.

The figure on the right shows how differents components interact on the demonstration computer.



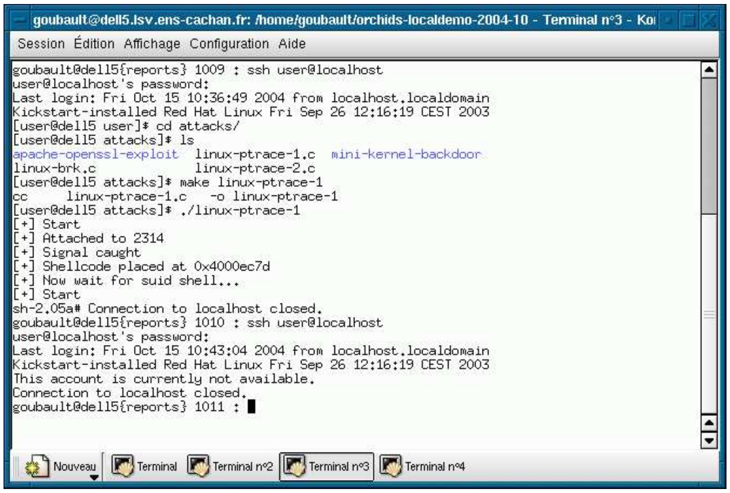

user@dell5:~/attacks - Terminal n°3 - Konsole
Session Édition Affichage Configuration Aide
Nouveau
Terminal
Terminal n°2
Terminal n°3
Terminal n°4

```
goubault@dell5{reports} 1007 : ssh user@localhost
user@localhost's password:
Last login: Fri Oct 15 10:35:52 2004 from localhost.localdomain
Kickstart-installed Red Hat Linux Fri Sep 26 12:16:19 CEST 2003
[user@dell5 user]$ cd attacks/
[user@dell5 attacks]$ ls
apache-openssl-exploit  linux-ptrace-1.c  mini-kernel-backdoor
linux-brk.c             linux-ptrace-2.c
[user@dell5 attacks]$ make linux-ptrace-1
cc     linux-ptrace-1.c   -o linux-ptrace-1
[user@dell5 attacks]$ ./linux-ptrace-1
[+] Start
[+] Attached to 2163
[+] Signal caught
[+] Shellcode placed at 0x4000ec7d
[+] Now wait for suid shell...
[+] Start
sh-2.05a# █
```

Fig. 3. Running a `ptrace` attack without ORCHIDS

The complete presentation of the tool consists in launching a few attacks, once with ORCHIDS disabled to show that the attacks really work and allow us to gain illegitimate `root` access; and a second time to show that the same attacks will be detected and countered by ORCHIDS. We do not require any specific equipment other than a videoprojector; we shall use a laptop of our own, with all necessary tools installed.

We illustrate just one attack here, corresponding to the one described in the paper. Figure 3 shows an intruder logging in through ssh on a machine (which will just be `localhost` in the demonstration), compiling a `ptrace` attack and getting root privileges. (As the `#` sign at the final prompt shows.) Note that the attack is particularly verbose, for pedagogical reasons, and explains every step it does: see the lines starting with `[+]`. (An actual attack would not be as verbose.)



goubault@dell5.lsv.ens-cachan.fr: /home/goubault/orchids-localdemo-2004-10 - Terminal n°3 - Ko
Session Édition Affichage Configuration Aide
Nouveau
Terminal
Terminal n°2
Terminal n°3
Terminal n°4

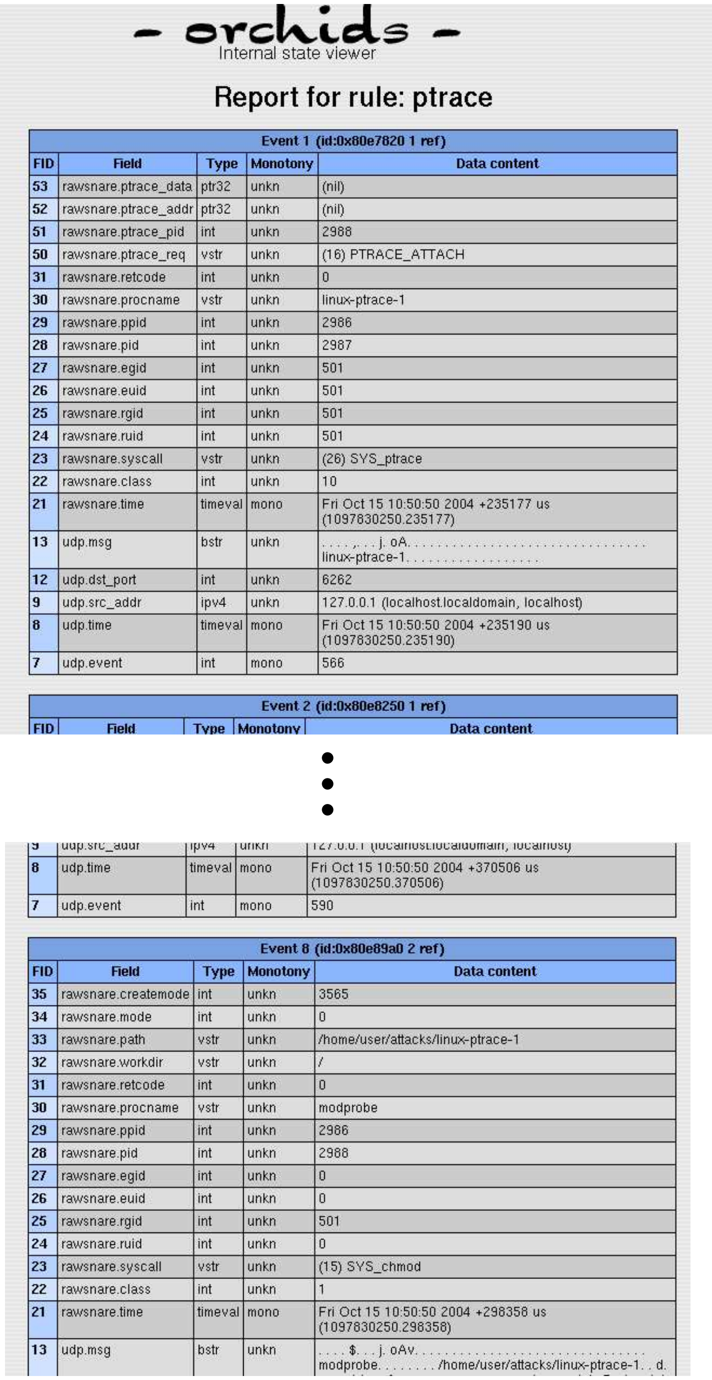
```
goubault@dell15{reports} 1009 : ssh user@localhost
user@localhost's password:
Last login: Fri Oct 15 10:36:49 2004 from localhost.localdomain
Kickstart-installed Red Hat Linux Fri Sep 26 12:16:19 CEST 2003
[user@dell15 user]$ cd attacks/
[user@dell15 attacks]$ ls
apache-openssl-exploit  linux-ptrace-1.c  mini-kernel-backdoor
linux-brk.c             linux-ptrace-2.c
[user@dell15 attacks]$ make linux-ptrace-1
cc     linux-ptrace-1.c   -o linux-ptrace-1
[user@dell15 attacks]$ ./linux-ptrace-1
[+] Start
[+] Attached to 2314
[+] Signal caught
[+] Shellcode placed at 0x4000ec7d
[+] Now wait for suid shell...
[+] Start
sh-2.05a# Connection to localhost closed.
goubault@dell15{reports} 1010 : ssh user@localhost
user@localhost's password:
Last login: Fri Oct 15 10:43:04 2004 from localhost.localdomain
Kickstart-installed Red Hat Linux Fri Sep 26 12:16:19 CEST 2003
This account is currently not available.
Connection to localhost closed.
goubault@dell15{reports} 1011 : █
```

Fig. 4. ORCHIDS counters the `ptrace` attack

When ORCHIDS is enabled, the same attack produces the response shown in Figure 4. The illegal connection is closed, and the account is closed (the intruder cannot log in again). The reason for the latter is that the attack actually succeeded, and there is a risk that the intruder managed to install a setuid bit shell, which would allow him to regain root privileges without even replaying an attack.

Figure 5 shows excerpts from the detailed report that ORCHIDS produces. Look at the first event that matched the attack (top). This is the call to `ptrace` (see field 23, `rawsnare.syscall`) using the `ATTACH` command (see field 50, `rawsnare.ptrace_req`) by process 2987 (field 28, `rawsnare.pid`) with effective uid 501 (field 24, `rawsnare.euid`) on process 2988 (field 51, `rawsnare.ptrace_pid`). The last event matching the attack, namely, the `DETACH` event is event 6 in the report (not shown).

Compared to the idealized formula (1), the formula contains additional transitions to track actions that *follow* the last characteristic event of the attack. Event 7 (not shown) in particular is a call from the (malicious) shellcode to `chown`, and event 8 (shown) is a call from the shellcode to `chmod`, indicating that the intruder managed to install the setuid bit on some (now root-owned) file before it got kicked out by ORCHIDS: the value of the protection bits (field 35, `rawsnare.createmode`) is decimal 3565, i.e. octal 6755, aka. `-rwsr-sr-x` in more readable Unix notation. This confirms our guess (above) that the intruder produced a setuid bit shell.



- orchids -
Internal state viewer

Report for rule: ptrace

Event 1 (id:0x80e7820 1 ref)				
FID	**Field**	**Type**	**Monotony**	**Data content**
53	rawsnare.ptrace_data	ptr32	unkn	(nil)
52	rawsnare.ptrace_addr	ptr32	unkn	(nil)
51	rawsnare.ptrace_pid	int	unkn	2988
50	rawsnare.ptrace_req	vstr	unkn	(16) PTRACE_ATTACH
31	rawsnare.retcode	int	unkn	0
30	rawsnare.procname	vstr	unkn	linux-ptrace-1
29	rawsnare.ppid	int	unkn	2986
28	rawsnare.pid	int	unkn	2987
27	rawsnare.egid	int	unkn	501
26	rawsnare.euid	int	unkn	501
25	rawsnare.rgid	int	unkn	501
24	rawsnare.ruid	int	unkn	501
23	rawsnare.syscall	vstr	unkn	(26) SYS_ptrace
22	rawsnare.class	int	unkn	10
21	rawsnare.time	timeval	mono	Fri Oct 15 10:50:50 2004 +235177 us (1097830250.235177)
13	udp.msg	bstr	unkn	. . . . ,. . . j. oA. . . . . linux-ptrace-1. . . . .
12	udp.dst_port	int	unkn	6262
9	udp.src_addr	ipv4	unkn	127.0.0.1 (localhost.localdomain, localhost)
8	udp.time	timeval	mono	Fri Oct 15 10:50:50 2004 +235190 us (1097830250.235190)
7	udp.event	int	mono	566

Event 2 (id:0x80e8250 1 ref)				
FID	**Field**	**Type**	**Monotony**	**Data content**

●
●
●

9	udp.src_addr	ipv4	unkn	127.0.0.1 (localhost.localdomain, localhost)
8	udp.time	timeval	mono	Fri Oct 15 10:50:50 2004 +370506 us (1097830250.370506)
7	udp.event	int	mono	590

Event 8 (id:0x80e89a0 2 ref)				
FID	**Field**	**Type**	**Monotony**	**Data content**
35	rawsnare.createmode	int	unkn	3565
34	rawsnare.mode	int	unkn	0
33	rawsnare.path	vstr	unkn	/home/user/attacks/linux-ptrace-1
32	rawsnare.workdir	vstr	unkn	/
31	rawsnare.retcode	int	unkn	0
30	rawsnare.procname	vstr	unkn	modprobe
29	rawsnare.ppid	int	unkn	2986
28	rawsnare.pid	int	unkn	2988
27	rawsnare.egid	int	unkn	0
26	rawsnare.euid	int	unkn	0
25	rawsnare.rgid	int	unkn	501
24	rawsnare.ruid	int	unkn	0
23	rawsnare.syscall	vstr	unkn	(15) SYS_chmod
22	rawsnare.class	int	unkn	1
21	rawsnare.time	timeval	mono	Fri Oct 15 10:50:50 2004 +298358 us (1097830250.298358)
13	udp.msg	bstr	unkn	. . . . $. . . j. oAv. . . . . modprobe. . . . . /home/user/attacks/linux-ptrace-1. . d.

Fig. 5. Report on the `ptrace` attack

A.2 Status

ORCHIDS is currently in a *prototype* status. It is deployed experimentally in the network of the laboratory where it was designed and implemented (LSV), consisting of about 150 equipments (servers, workstations and routers). It supports common standards to make deployment easier (Unix syslog, sendmail, apache logs, snmp protocol, etc.) ORCHIDS is designed so as to run as autonomously as possible. Naturally, it still requires a systems or security administrator.

It has been deployed for local, real-conditions testing on the network of the lab since September, 2004. We do not want to publicize this yet, as this kind of announcement invariably attracts script kiddies. Since ORCHIDS is still a prototype, this would only augment the chances that our network be hacked into.

A.3 Availability

ORCHIDS is not available in source form. The single-laptop demonstration version is however available free of charge for academic use only, in binary form (compiled for Linux with glibc; current version is known to work on Linux 2.4.18-3, 2.4.20-20.7, and 2.4.20-24.7), upon request to the authors. This is subject to a specific licence.

Installation requires root privileges (for, e.g., modifying certain configuration files, and installing the ORCHIDS in-kernel audit daemon). Additionally, to detect `ptrace` attacks, the Snare in-kernel audit module should be installed, too. Several past versions of Snare are known not to report some of the events that we require, e.g., calls to `execve`. To get around this kind of problem, the current version of ORCHIDS uses a hacked-up version of Snare, which additionally reports system events in binary format (while Snare outputs formatted text), for added efficiency.

The attacks demonstrated are known to work on Linux 2.4.18-3, Red Hat release 7.3 (Valhalla). The `ptrace` attack will *not* work on higher versions. In general, the demonstrated versions will *not* work on up-to-date versions of Linux: we will not demonstrate attacks that have not been countered already.

The `mod_ssl` attack (a network attack allowing an intruder to gain access to a remote computer, which we shall demonstrate as a preparation for launching a `ptrace` attack; not described here for space reasons) requires the Apache Web server v.1.3.23-11 (exactly), the `mod_ssl-2.8.7-4` Apache SSL extension (exact version again), and the cryptographic package `openssl-0.9.6b-18` (again, exact version).

We do not wish to distribute the attacks, whether in source or binary code. While there is no jurisprudence yet, distributing the attacks might be interpreted as a felony on our part under recent French laws (specifically, the LEN, i.e., the “Loi sur l’Electronique Numérique”, Loi 2004-575 of June 21, 2004, see `http://www.iris.sgdg.org/actions/len/`).

See the Web page `http://www.lsv.ens-cachan.fr/orchids/` for more information on ORCHIDS.