Automated Fault Identification

Hashem Waly
Béchir Ktari

Département d'informatique et de génie logiciel
Faculté des Sciences et de Génie
Université Laval, Québec, Canada

December 10, 2009
Montréal, Canada

Agenda

Agenda

Agenda

Motivation

- Malicious activities, performance bottlenecks, and debugging important events in a system are kinds of behaviors that are essential in the surveillance and maintaining the reliability of large systems.
- The detection of such behaviors has become a more challenging task with the emergence of multi-core, multi-threaded processes and the higher level of inter connectivity (between networked systems).

Objectives

- **Automating** the detection of malicious behaviors, performance degradation, and software bugs, in the context of multi-core/multi-processor CPUs, and distributed systems.
- Avoid to affect the performance of the system being analyzed.
- Integration within a software development environment (*Eclipse*).

Intrusion Detection Systems (IDS)

Based on the type of intrusions:

- Host-based
- Network-based

Based on the detection techniques:

- Signature-Based
- Anomaly-Based
- Intermediate approach (Policy-Based)

Based on the detection engine:

- On-line
- Off-line

Signature-Based (IDS)

Antivirus Behaviors

Search for the occurrence of a specific set of characters (usually) at the beginning of files.

Rule-Based

Specify one rule for describing attacks.

Scenario-Based

Abstract the attack to be a scenario composed from a set of high level events.

Policy-Based

Detects anomalies that violate policy rules rather than a learned behavior considered to be "normal".

Signature-Based (IDS)

Antivirus Behaviors

Search for the occurrence of a specific set of characters (usually) at the beginning of files.

Rule-Based

Specify one rule for describing attacks.

Scenario-Based

Abstract the attack to be a scenario composed from a set of high level events.

Policy-Based

Detects anomalies that violate policy rules rather than a learned behavior considered to be "normal".

Signature-Based (IDS)

Antivirus Behaviors

Search for the occurrence of a specific set of characters (usually) at the beginning of files.

Rule-Based

Specify one rule for describing attacks.

Scenario-Based

Abstract the attack to be a scenario composed from a set of high level events.

Policy-Based

Detects anomalies that violate policy rules rather than a learned behavior considered to be "normal".

Signature-Based (IDS)

Antivirus Behaviors

Search for the occurrence of a specific set of characters (usually) at the beginning of files.

Rule-Based

Specify one rule for describing attacks.

Scenario-Based

Abstract the attack to be a scenario composed from a set of high level events.

Policy-Based

Detects anomalies that violate policy rules rather than a learned behavior considered to be "normal".

Milestones

K1.1

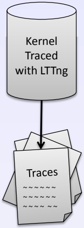
Build a list of low level problems and collect a database of good traces and of traces illustrating typical problems.

K1.2

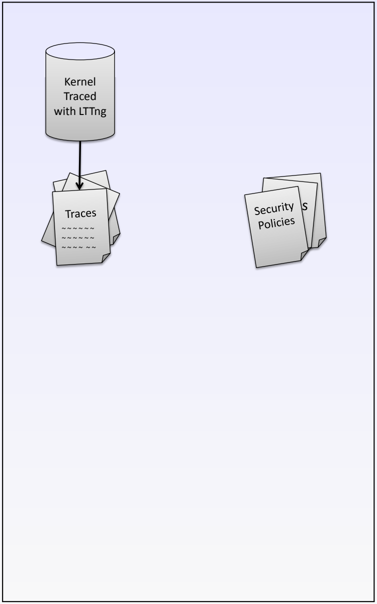
Study various languages that may be suitable to describe different fault patterns. Compare their expressiveness, potential for performance, and applicability to detect a wide range of problems.

Agenda

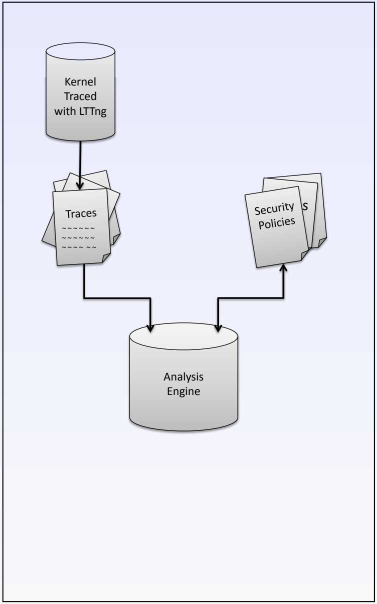
Architecture



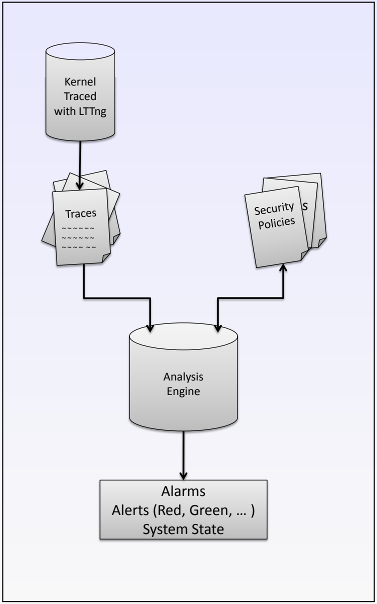
Architecture



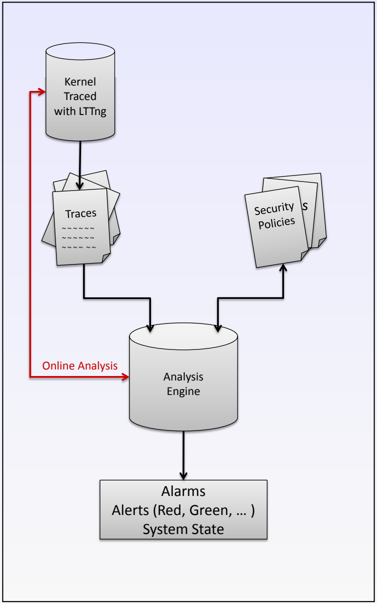
Architecture



Architecture

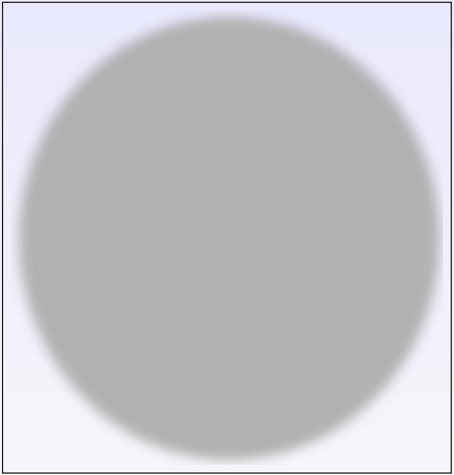


Architecture



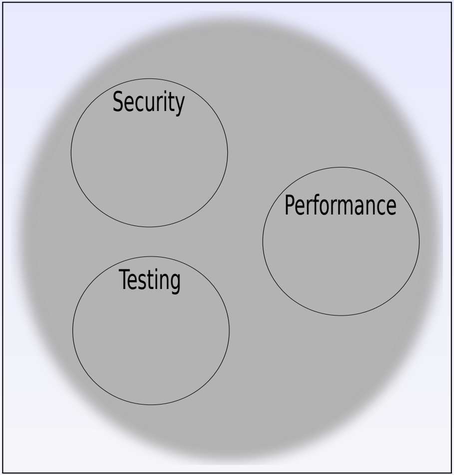
Agenda

What kind of problems?



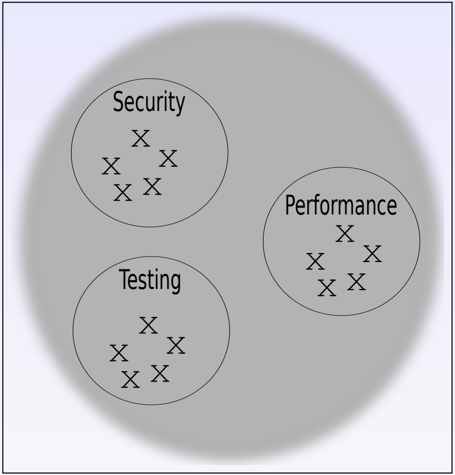
- Very wide range of problems.
- **Refine** The problems into three groups.
- **Select** a set of problems for each group:
  - Clients requirements.
  - Domain classics (Buffer overflow, deadlock, ...).
  - Litterature review.

What kind of problems?



- Very wide range of problems.
- **Refine** The problems into three groups.
- **Select** a set of problems for each group:
  - Clients requirements.
  - Domain classics (Buffer overflow, deadlock, ...).
  - Litterature review.

What kind of problems?

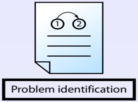


- Very wide range of problems.
- **Refine** The problems into three groups.
- **Select** a set of problems for each group:
  - Clients requirements.
  - Domain classics (Buffer overflow, deadlock, ...).
  - Litterature review.

Agenda

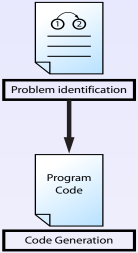
Scientific Model

- Problem Description.
- Problem Generation (code re-use, or self-development)
- LTTng Trace analysis (refine the trace and study relevant events).
- Study Good Traces.
- Language properties.
- Analysis (alternate attacks, solutions, references to threads Databases).



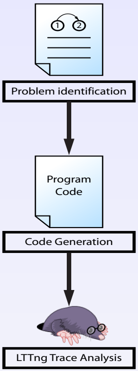
Scientific Model

- Problem Description.
- Problem Generation (code re-use, or self-development)
- LTTng Trace analysis (refine the trace and study relevant events).
- Study Good Traces.
- Language properties.
- Analysis (alternate attacks, solutions, references to threads Databases).



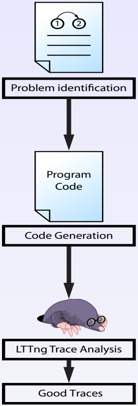
Scientific Model

- Problem Description.
- Problem Generation (code re-use, or self-development)
- LTTng Trace analysis (refine the trace and study relevant events).
- Study Good Traces.
- Language properties.
- Analysis (alternate attacks, solutions, references to threads Databases).



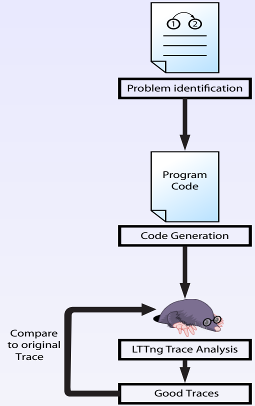
Scientific Model

- Problem Description.
- Problem Generation (code re-use, or self-development)
- LTTng Trace analysis (refine the trace and study relevant events).
- Study Good Traces.
- Language properties.
- Analysis (alternate attacks, solutions, references to threads Databases).



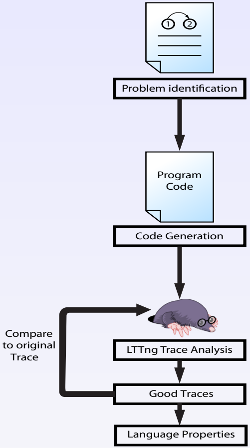
Scientific Model

- Problem Description.
- Problem Generation (code re-use, or self-development)
- LTTng Trace analysis (refine the trace and study relevant events).
- Study Good Traces.
- Language properties.
- Analysis (alternate attacks, solutions, references to threads Databases).



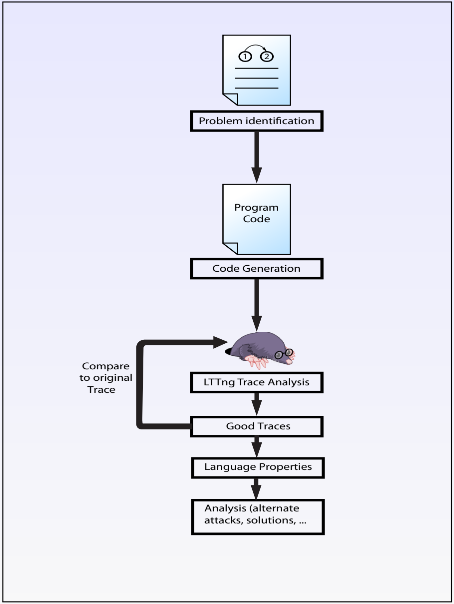
Scientific Model

- Problem Description.
- Problem Generation (code re-use, or self-development)
- LTTng Trace analysis (refine the trace and study relevant events).
- Study Good Traces.
- Language properties.
- Analysis (alternate attacks, solutions, references to threads Databases).



Scientific Model

- Problem Description.
- Problem Generation (code re-use, or self-development)
- LTTng Trace analysis (refine the trace and study relevant events).
- Study Good Traces.
- Language properties.
- Analysis (alternate attacks, solutions, references to threads Databases).



Agenda

Malicious Traces

Security

- File permissions and attributes.
  - Escaping a chroot Jail.
  - Race conditions on files.
- Privilege Escalation.
  - Abusing `setuid` function.
- Buffer Overflow.
- Networks.
  - SYN Flood attack.
- Viruses.
  - Virus installation
  - Linux RST.b virus.

Testing Programs

- Using File Descriptors
- Deadlock
- Error-Handling

System Performance

- Inefficient I/O
- Real-time Applications
- Excessive Swapping

Malicious Traces

Security

- File permissions and attributes.
  - Escaping a chroot Jail.
  - Race conditions on files.
- Privilege Escalation.
  - Abusing `setuid` function.
- Buffer Overflow.
- Networks.
  - SYN Flood attack.
- Viruses.
  - Virus installation
  - Linux RST.b virus.

Testing Programs

- Using File Descriptors
- Deadlock
- Error-Handling

System Performance

- Inefficient I/O
- Real-time Applications
- Excessive Swapping

Malicious Traces

Security

- File permissions and attributes.
  - Escaping a chroot Jail.
  - Race conditions on files.
- Privilege Escalation.
  - Abusing `setuid` function.
- Buffer Overflow.
- Networks.
  - SYN Flood attack.
- Viruses.
  - Virus installation
  - Linux RST.b virus.

Testing Programs

- Using File Descriptors
- Deadlock
- Error-Handling

System Performance

- Inefficient I/O
- Real-time Applications
- Excessive Swapping

Malicious Traces

Security

- File permissions and attributes.
  - Escaping a chroot Jail.
  - Race conditions on files.
- Privilege Escalation.
  - Abusing `setuid` function.
- Buffer Overflow.
- Networks.
  - SYN Flood attack.
- Viruses.
  - Virus installation
  - Linux RST.b virus.

Testing Programs

- Using File Descriptors
- Deadlock
- Error-Handling

System Performance

- Inefficient I/O
- Real-time Applications
- Excessive Swapping

Race Conditions on files - Problem Description

Problem

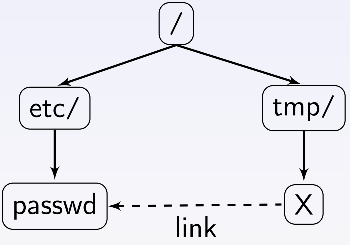
Race Condition occurs when a system or a device assumes to perform two or more operations atomically while they are not.

Race condition on files

Binding the name to an object changes between repeated references occur in many programs

Race Conditions on files - Problem Generation

```
if (access("/tmp/X", W_OK) == 0) {
    unlink("/tmp/X");
    symlink("/etc/passwd","/tmp/X");
    if((fd = open("/tmp/X", O_WRONLY)) == -1) {
        perror("/tmp/X");
        return(0);
    }
}
```



Race Conditions on files - Problem Generation

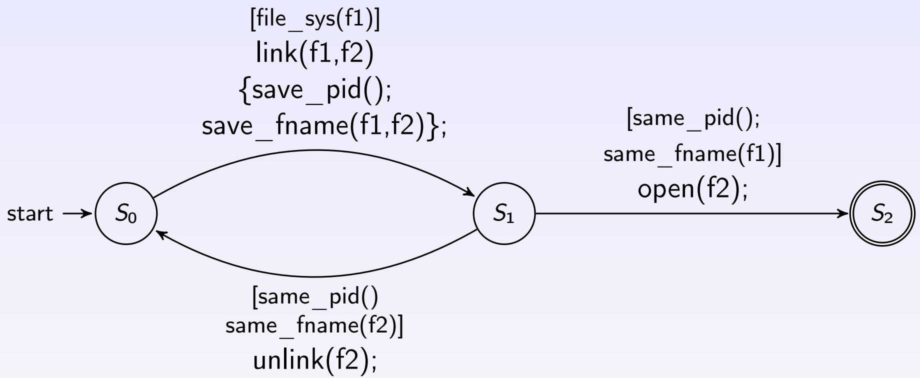


Figure: Race condition FSM

Race Condition on files - LTTng Trace

```
kernel.syscall_entry: 105.11303 (./kernel_1), 7914, 7914, ./race_violation,
31269, 0x0, SYSCALL { ip = 0xb7fc3430, syscall_id = 33 [sys_access+0x0/0x30]}

kernel.syscall_exit: 1305.36650(./kernel_1), 7914, 7914, ./race_violation,
31269, 0x0, USER_MODE { ret = 0 }

kernel.syscall_entry: 1305.01835 (./kernel_1), 7914, 7914, ./race_violation,
31269, 0x0, SYSCALL { ip = 0xb7fc3430, syscall_id = 10 [sys_unlink+0x0/0x20] }

kernel.syscall_exit: 1305.35302 (./kernel_1), 7914, 7914, ./race_violation,
31269, 0x0, USER_MODE { ret = 0}

kernel.syscall_entry: 1305.85091 (./kernel_1), 7914, 7914, ./race_violation,
31269, 0x0, SYSCALL {ip = 0xb809a430, syscall_id = 83[sys_symlink+0x0/0x30]}

kernel.syscall_exit: 1305.35302 (./kernel_1), 7914, 7914, ./race_violation,
31269, 0x0, USER_MODE { ret = 0 }

kernel.syscall_entry: 1305.28196 (./kernel_1), 7914, 7914, ./race_violation,
31269, 0x0, SYSCALL { ip = 0xb809a430, syscall_id = 5 [sys_open+0x0/0x40]}
```

Figure: LTTng trace of Race conditions on files

Race Conditions on files - Language properties

The properties of the language are summarized as follows:

1. **Scenario based on multiple events**
2. **Conditional Transitions**
3. **Variables**
4. **Grouping**
5. **Real-time Constraints**

Race Conditions on files - Good traces

```
if (access(filename, W_OK) == 0) {
    if((fd = open(filename, O_WRONLY)) == -1) {
        perror(filename);
        return(0);
    }
    write(fd,"hello\n",6); //write to file
    close(fd);
}
```

Race Conditions on files - Discussion

Real security threads

- Kumar and Spafford gain root access on unix systems using the superuser's privileges.

S. Kumar and E. Spafford.
"An Application of Pattern Matching in Intrusion Detection".
Technical Report 94–013, Department of Computer Science, Purdue University, 1994.

- Bishop and Dilger, on SunOS and HP/UX systems, gain root access by interleaving the operation of passwd process.

M .Bishop and M .Dilger.
"Checking for Race Conditions in File Accesses".
The USENIX Association, Computing Systems, Vol. 9, No. 2, pp. 131–152, 1996.

- Advisory-5.UNIX.mail `Binmail` race condition.

Abusing setuid - Problem Description

Problem

Privilege Escalation It's the act of exploiting a bug or design flaw in a software application to gain access to resources which normally would have been protected from an application or user.

If the process runs with elevated privileges, it's not secure to execute risky activities:

openning a shell with administrator privileges, ...

D. Dean, H. Chen and D. Wagner.
"Model checking one million lines of C code".
Proceedings of the 11th Annual Network and Distributed System Security Symposium, 2004.

Abusing setuid - Problem Description

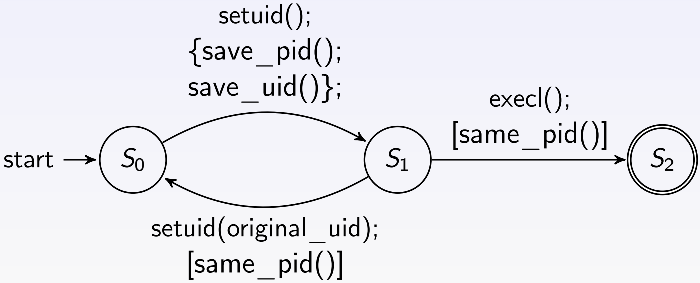
Problem

Privilege Escalation It's the act of exploiting a bug or design flaw in a software application to gain access to resources which normally would have been protected from an application or user.

If the process runs with elevated privileges, it's not secure to execute risky activities:

openning a shell with administrator privileges, ...

D. Dean, H. Chen and D. Wagner.
"Model checking one million lines of C code".
Proceedings of the 11th Annual Network and Distributed System Security Symposium, 2004.



Abusing setuid - Problem Generation & LTTng Trace

```
setuid(0);
execl("/bin/sh","/bin/sh",NULL);
```

```
kernel.syscall_entry: 595.3171 (./Trace/kernel_0), 20223, 20223, ./Privilege
20222, 0x0, SYSCALL { ip = 0xb8068430, syscall_id = 213 [sys_setuid+0x0/0xe0]}
kernel.syscall_exit: 595.06439 (./Trace/kernel_0), 20223, 20223, ./Privilege
20222, 0x0, USER_MODE { ret = 0}

kernel.syscall_entry: 595.71835 (./Trace/kernel_0), 20223, 20223, ./Privilege
20222, 0x0, SYSCALL { ip = 0xb8068430, syscall_id = 11 [ptregs_execve+0x0/0x10] }
fs.exec: 18595.918502002 (./Trace/fs_0), 20223, 20223, /bin/sh
20222, 0x0, SYSCALL filename = "/bin/sh"
kernel.syscall_exit: 595.03611 (./Tracekernel_0), 20223, 20223, /bin/sh,
20222, 0x0, USER_MODE { ret = 0 }
```

Abusing setuid - Customizing LTTng trace

LTTng provides by default a general information about the events running on the system.

1. *Enabling/Disabling* markers (which represents the type of events printed in trace files).
2. Customizing the *information* provided about the events (like the *Event Parameters*).
   - TracePoints
   - Markers

Abusing setuid - Customizing LTTng trace (continued)

1. **Find the implementation** ("./kernel/sys.c").
2. **Define the marker**
3. **Compile**
4. **Trace**

```
SYSCALL_DEFINE1(setuid, uid_t, uid)
{
    ...
    trace_mark(kernel, syscall_setuid, "UID %d", uid);
    ...
}
```

Abusing setuid - Customizing LTTng trace (continued)

```
kernel.syscall_entry: 595.03171 (./Trace/kernel_0), 20223, 20223,./Privilege
20222, 0x0, SYSCALL { ip = 0xb8068430, syscall_id = 213 [sys_setuid+0x0/0xe0] }

kernel.syscall_setuid: 595.01131 (./Trace/kernel_0), 20223, 20223,./Privilege
20222, 0x0, SYSCALL { UID = 1000 }

kernel.syscall_exit: 595.06439 (./Trace/kernel_0), 20223, 20223, ./Privilege
20222, 0x0, USER_MODE { ret = 0}
```

Abusing setuid - Good Traces

- Change the user ID of the process.
- Perform privilege capabilities.
- Return to the normal user ID.

```
if( setuid(0) == -1 ){
   printf("ERROR: %s\n",strerror(errno));
}
//execute the commands as privileged users
//chroot for example
chroot("/home/hamow1/myjail");
chdir("/");
//return to your user ID
if( setuid(original uid) == -1 ){
   printf("ERROR: %s\n",strerror(errno));
}
```

Good Traces

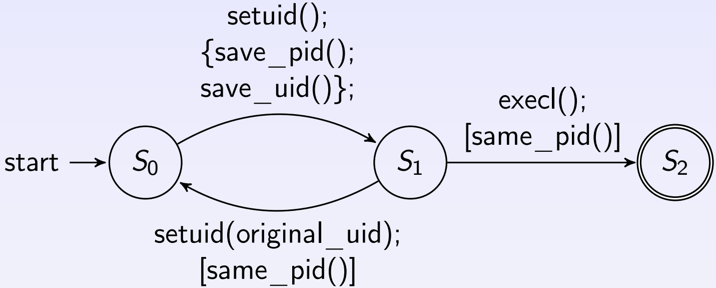
```
kernel.syscall_entry:178828.081389920(./Trace/kernel\_0), 24896, 24896, ./priv,
5870, 0x0, SYSCALL { ip = 0xb7f23430, syscall_id = 213 [sys_setuid+0x0/0xe0] }
kernel.syscall_exit:178828.081391660(./Trace/kernel_0), 24896, 24896, ./priv,
5870, 0x0, USER_MODE { ret = 0 }

kernel.syscall_entry:178828.081395540(./Trace/kernel_0), 24896, 24896, ./priv
5870, 0x0, SYSCALL { ip = 0xb7f23430, syscall_id = 61 [sys_chroot+0x0/0xa0]}
kernel.syscall_exit:178828.918503611(./Trace/kernel_0), 24896, 24896, ./priv
5870, 0x0, USER_MODE { ret = 0}

kernel.syscall_entry:178828.081401405(./kernel_0), 24896, 24896, ./priv
5870, 0x0, SYSCALL { ip = 0xb7f99430, syscall_id = 12 [sys_chdir+0x0/0x80]}
kernel.syscall_exit:178828.081402218(./kernel_0), 24896, 24896,./priv,
5870, 0x0, USER_MODE { ret = 0 }

kernel.syscall_entry:178828.081402849(./Trace/kernel_0), 24896, 24896, .\-/priv
5870, 0x0, SYSCALL { ip = 0xb7f23430, syscall_id = 213 [sys_setuid+0x0/0xe0]}
kernel.syscall_exit:178828.081405461(./Trace/kernel_0), 24896, 24896, ./priv,
5870, 0x0, USER_MODE { ret = 0 }
```

Language Properties



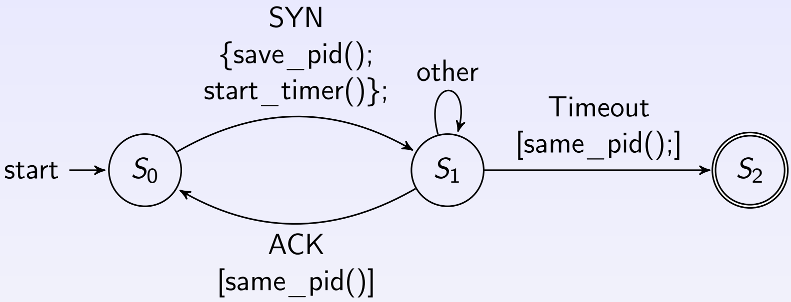
1. Scenario based on multiple events.
2. Conditional Transitions.
3. Variables.
4. Grouping.

SYN Flood attack - Problem Description

Problem

The SYN flood attack is a denial of service attack that consists in flooding a server with half-open TCP connections. Once all resources set aside for half-open connections are reserved, no new connections (legitimate or not) can be made, resulting in denial of service.

SYN Flood attack - Problem Generation




```
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
connect(sock, (struct sockaddr *) &echoServAddr,
        sizeof(echoServAddr));
send(sock, echoString, echoStringLen, 0) != echoStringLen);
//bytesRcvd = recv(sock,echoBuffer,RCVBUFSIZE-1,0);
```

SYN Flood attack - LTTng Trace

```
kernel.syscall_entry: 597.42938 (./Trace/kernel_0), 15731, 15731, ./flood,
14070, 0x0, SYSCALL {ip = 0xb7fd2430, syscall_id=102 [sys_socketcall+0x0/0x320]}
net.socket_call: 754.66029 (./Trace/net_0), 15731, 15731, ./flood,
14070, 0x0, SYSCALL { call = 1, a0 = 2 }
net.socket_create}: 597.61169 (./Trace/net_0), 15731, 15731, ./flood,14070,
0x0, SYSCALL { family = 2, type = 1, protocol = 6, sock = 0xf17e5800, ret = 3 }
kernel.syscall_exit: 597.61926 (./Trace/kernel_0), 15731, 15731, ./flood,
14070, 0x0, USER_MODE { ret = 3 }

kernel.syscall_entry: 597.73730 (./Trace/kernel_0), 15731, 15731, ./flood,14070,
0x0, SYSCALL { ip = 0xb7fd2430, syscall_id = 102 [sys_socketcall+0x0/0x320] }
net.socket_call: 597.74407 (./Trace/net_0), 15731, 15731, ./flood,
14070, 0x0, SYSCALL { call = 3, a0 = 3}
net.socket_connect: 597.89308 (./Trace/net_0), 15731, 15731, ./flood,
14070, 0x0, SYSCALL { fd = 3, uservaddr = 0xbf868b04, addrlen = 16, ret = 0 }
kernel.syscall_exit: 597.89498 (./Trace/kernel_0), 15731, 15731, ./flood,
14070, 0x0, USER_MODE { ret = 0 }

kernel.syscall_entry: 597.85438 (./Trace/kernel_0), 15731, 15731, ./flood,14070,
0x0, SYSCALL { ip = 0xb7fd2430, syscall_id = 102 [sys_socketcall+0x0/0x320]}
net.socket_call: 597.99178 (./Trace/net_0), 15731, 15731, ./flood,
14070, 0x0, SYSCALL { call = 9, a0 = 3 }
kernel.syscall_exit: 597.94148 (./Trace/kernel_0), 15731, 15731, ./flood,
14070, 0x0, USER_MODE { ret = 5 }
```

SYN Flood attack - Language properties

1. **Scenario based on multiple events**
2. **Conditional Transitions**
3. **Variables**
4. **Grouping**
5. **Real-time Constraints**
6. **Synthetic Events**

SYN Flood attack - Good traces

```
sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
connect(sock, (struct sockaddr *) &echoServAddr,
        sizeof(echoServAddr));
send(sock, echoString, echoStringLen, 0) != echoStringLen);
bytesRcvd = recv(sock,echoBuffer,RCVBUFSIZE-1,0);
```

SYN Flood attack - Discussion

Different network attacks

- **Sniffing**
- **Brute Force**
- **Buffer Overflows**
- **Spoofing**
- **Flooding**

An attack that is sensitive in time between events (Real-time constraints).
Could be used to describe more attacks.

Agenda

Deadlock - Problem Description

Problem

A **deadlock** is a situation wherein two or more competing actions are waiting for the other to finish, and thus neither ever does.

Locking Validator

For validating the occurence of deadlock in programs but there exists a lot of cases that is difficult to detect

Deadlock - Generation

PID	holding_ID	waiting_for_ID
1	a	b
3	d	.
2	b	a

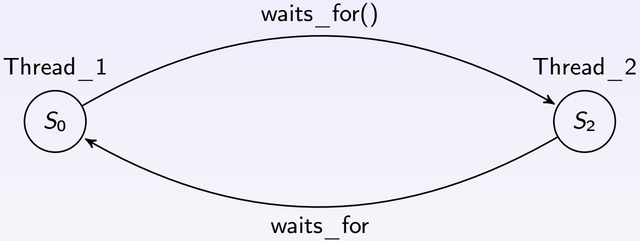
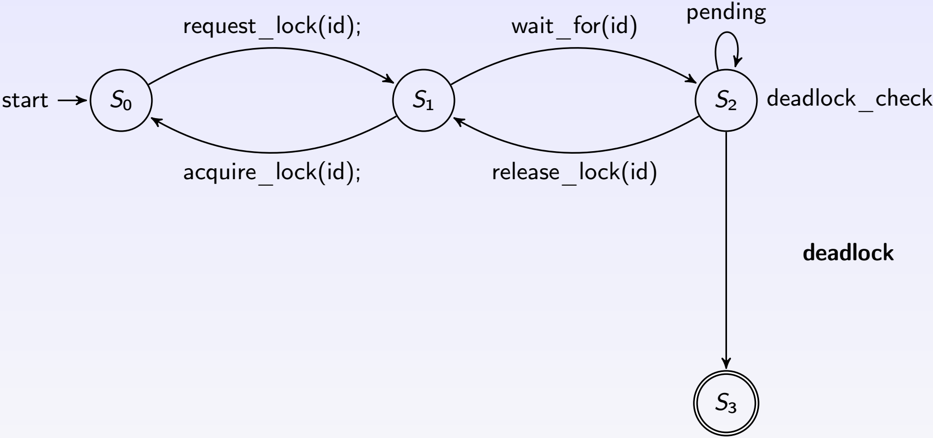


Figure: Race condition FSM

Deadlock - Problem Description



Deadlock - Problem Generation

```
void *function1();
void *function2();
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutex2 = PTHREAD_MUTEX_INITIALIZER;
int main()
{
   int rc1, rc2;
   pthread_t thread1, thread2;
   if( (rc1=pthread_create( &thread1, NULL, &function1, NULL)) )
      printf("Thread creation failed: %d\n", rc1);
   if( (rc2=pthread_create( &thread2, NULL, &function2, NULL)) )
      printf("Thread creation failed: %d\n", rc2);
   pthread_join( thread1, NULL);
   pthread_join( thread2, NULL);
   exit(0);
}
void *function1()
{
   pthread_mutex_lock( &mutex1 );
   pthread_mutex_lock( &mutex2 );
   pthread_mutex_unlock(&mutex2);
   pthread_mutex_unlock(&mutex1);
}
void *function2()
{
   pthread_mutex_lock( &mutex2 );
   pthread_mutex_lock( &mutex1 );
   pthread_mutex_unlock(&mutex1);
   pthread_mutex_unlock(&mutex2);
}
```

Deadlock - Language properties

- **Scenario based on multiple events**
- **Conditional Transitions**
- **Variables**
- **Grouping**
- **Global structure**
- **Real-time constraints**

Agenda

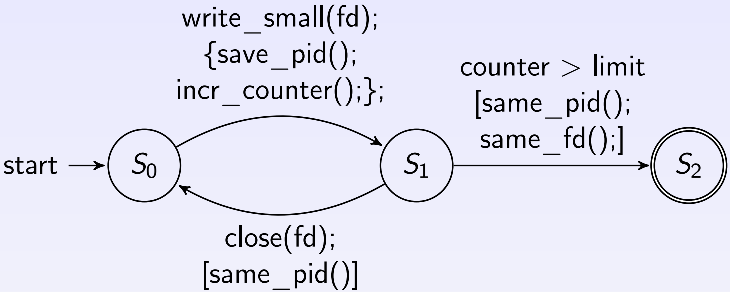
Inefficient I/O - Problem Description

- Inefficient I/O
  - Frequent writing of small chuncks of data.
  - Writing latency (timeout) to disk (maybe due to disk saturation, ...).
  - Reading twice the same data.
  - Reading the data that has been just written to disk.
- Real-time applications constraints.

Inefficient I/O - Problem Description

- Inefficient I/O
  - Frequent writing of small chuncks of data.
  - Writing latency (timeout) to disk (maybe due to disk saturation, ...).
  - Reading twice the same data.
  - Reading the data that has been just written to disk.
- Real-time applications constraints.

Inefficient I/O - Problem Description




```
fd = open("/home/hashem/test2.txt", O_RDONLY);
for(i=0;i<100;i++){
   write(fd,"a",1);
}
```

Inefficient I/O - LTTng Trace Details

1. kernel.syscall_entry: 103158.477573944 (./Trace/kernel_0), 19035, 19035, ./perf, , 17201, 0x0,
   USER_MODE ip = 0xb7f6d430, syscall_id = 5 [sys_open+0x0/0x40]
2. fs.open: 103158.47758115 (./Trace/fs_0), 19035, 19035, ./perf, , 17201, 0x0, SYSCALL
   fd = 3, filename = "/home/hashem/test2.txt"
3. kernel.syscall_exit: 103158.477582114 (./Trace/kernel_0), 19035, 19035, ./perf, , 17201, 0x0,
   USER_MODE ret = 3
4. kernel.syscall_entry: 103158.477582836 (./Trace/kernel_0), 19035, 19035, ./perf, , 17201, 0x0,
   USER_MODE ip = 0xb7f6d430, syscall_id = 4 [sys_write+0x0/0xb0]
5. fs.write: 103158.477637465 (./Trace/fs_0), 19035, 19035, ./perf, , 17201, 0x0, SYSCALL
   count = 1, fd = 3
6. kernel.syscall_exit: 103158.477582114 (./Trace/kernel_0), 19035, 19035, ./perf, , 17201, 0x0,
   USER_MODE ret = 1

Agenda

Discussion

1. Scenario based on multiple events.
2. Conditional Transitions.
3. Variables.
4. Grouping.
5. Counting.
6. Real-time constraints.
7. Non-Occurrence of events.
8. Synthetic events.

Name	1	2	3	4	5	6	7	8
chroot jail	✓	✓	✓	✓	-	-	-	-
Abusing setuid	✓	✓	✓	✓	-	-	-	-
Race condition	✓	✓	✓	✓	-	-	✓	-
SYN Flood	✓	✓	✓	✓	✓	✓	-	✓
File descriptors	✓	✓	✓	✓	-	-	-	-
Deadlock	✓	✓	✓	✓	-	-	-	-
Error Handling	✓	✓	✓	✓	-	-	-	-
Inefficient I/O	✓	✓	✓	✓	✓	-	-	✓

Agenda

Agenda

Scientific Model

- Wide range of systems.
- Group the languages used in these systems into the following groups:
  1. Imperatives Languages.
  2. Automata-based Languages.
  3. Temporal Logic.
  4. Expert Systems.
  5. Policy-based Languages.
  6. Other languages.
- Study each language using a scientific model.

Scientific Model

- Language Description (capabilities, syntax, ...).
- Example of property.
- Language Analysis:
  - Expressiveness.
  - Unambiguous.
  - Online/Offline.
  - Simplicity.
  - Trace dependency.
  - Open Source availability.

Agenda

Scenario Description Languages

Automating the detection of faulty behavior needs a simple and unambiguous language.

The languages are divided into the following categories:

1. Imperatives Languages (RUSSEL, BRO, DTrace, and SystemTap).
2. Automata-based Languages (STATL, State Machine Compiler, Ragel, BSML, and IDIOT).
3. Temporal Logic (ADele, Chronicle, and LogWeaver).
4. Expert Systems (P-Best, and Lambda).
5. Policy-based Languages (Blare, and BlueBox).
6. Other languages (snort, and SECnology).

Imperative Languages: RUSSEL (2006)

- RUle-baSed Sequence Evaluation Language.
- Used in audit trace analysis as part of ASAX IDS.

A. Mounji, N. Habra1, B. Le Charlier and I. Mathieu.
"Asax: Software architecture and rule-based language for universal audit trail analysis".
Computer Security – ESORICS 92, 648/1992:435–450, April 2006.

```
rule Failed_login (maxtimes, duration :  integer)
begin
if evt='login' and res='failure' and is_unsecure (terminal)
    -> Trigger off for next Count_rule1 (maxtimes-1, timestp+duration)
fi;
Trigger off for next Failed_login (maxtimes, duration)
end;
```

RUSSEL Rule

```
rule Count_rule1 (countdown, expiration :  integer)
if evt='login' and res='failure'
   and is_unsecure(terminal) and timesto < expiration
    -> if countdown > 1
        -> Trigger off for next Count_rule1(countdown-1, expiration);
           countdown=1
        -> SendMessage("too much failed login's")
        fi;
     timestp >= expiration
     -> Skip;
     true
     -> Trigger off for next Count_rule1(countdown, expiration);
fi;
```

RUSSEL Discussion

- Expressiveness: Provide more flexibility to describe attacks than declarative languages (one rule-based languages).
- Unambiguous.
- Online/Offline: Online.
- Simplicity: Difficult to describe complex attacks.
- Trace dependency: Dependent on Network attacks.

Automata-Based Languages: STATL (2002)

- Language used in STAT for IDS.
- STATL is translated into C++.
- Contains a lot of extensions like: NetStat, WinStat, LinStat, ... (Contains a set of pre-defined scenarios)
- Visualization tool could be used.
- Provide Timers.

S.T. Eckmann, G. Vigna, and R.A. Kemmerer.
"STATL: An Attack Language for State-based Intrusion Detection".
Journal of Computer Security, vol. 10, no. 1/2, pp. 71-104, 2002.

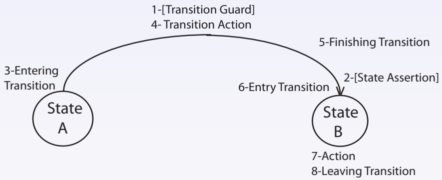


Figure: General Finite State Machine Architecure

Automata-Based Languages: STATL (2002)

- Three types of transitions:
  1. Consuming: Normal transition, system changes it state.
  2. Non-Consuming: Create a copy of the system state, and then moves to next state.
  3. Unwinding: Delete all states.

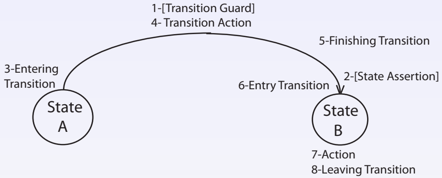


Figure: General Finite State Machine Architecure

Scenario Example - FSM

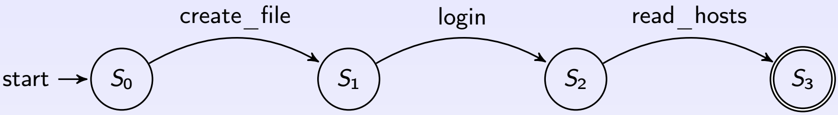


Figure: ftp-write FSM

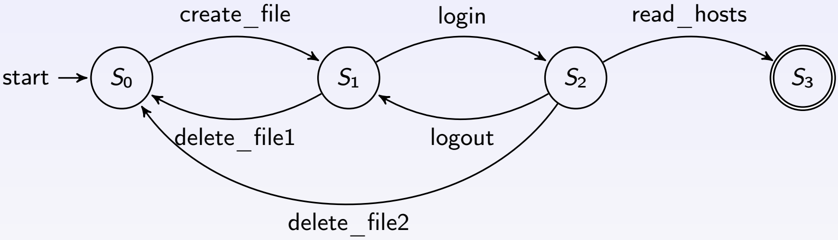


Figure: State transition diagram with unwinding transitions

Scenario Example

```
use ustat;
scenario ftp_write
{
        int user;
        int pid;
        int inode;

        initial state s0 {}
        transition create_file (s0 -> s1) nonconsuming
        {
                [WRITE w] : (w.euid != 0) && (w.owner != w.ruid))
                { inode = w.inode; }
        }
        state s1 {}
        transition login (s1 -> s2) nonconsuming
        {
                [EXEC e] : match_name(e.objname, "login")
                { user = e.ruid; pid = e.pid; }
        }
        state s2 {}
        transition read_rhosts (s2 -> s3) consuming
        {
                [READ r] : (r.pid == pid) && (r.inode == inode)
        }
        state s3
        {
                {
                        string username;
                        userid2name(user, username);
                        log("remote user %s gained local access", username);
                }
        }
```

Scenario Example - Unwinding transitions

```
        transition delete_file1 (s1 -> s0) unwinding
        {
                [DELETE d] : d.inode == inode
        }
        transition delete_file2 (s2 -> s0) unwinding
        {
                [DELETE d] : d.inode == inode
        }
        transition logout (s2 -> s1) unwinding
        {
                [EXIT e] : e.pid == pid
        }
}
```

STATL - Discussion

- Expressiveness: A flexible way for describing attacks in a form that could be represented graphically, the conversion of the code to C++ make it more expressive.
- Unambiguous.
- Online/Offline: Online.
- Simplicity: Simple way to describe complex attacks.
- Trace dependency: Contains different packages for different problems (WinStat, UStat, NetStat).

Temporal Logic: Chronicle

- Temporal logic that permits the recognition of chronics in a flow of events.
- Verified by the online system "Chronicle Recognition System".

C. Dousson.
"Suivi d'évolutions et reconnaissance de chroniques".
Ph.D. dissertation, Universitï£¡ Paul Sabatier de Toulouse, september 1994.

- Chronicle operators:
  - **hold(P;v;(t;t2))**: The attribute P holds the value of v, in the interval t1 to t2.
  - **event(P;(v1;v2);):** the attribute P changes from value v1 to v2 in time t.
  - **event(P;t)**: The attribute p occurs at time t.
  - **noevent(P;(t1;t2))**: The value of attribute P has not changed in the interval (t1,t2)
  - **occurs((n1;n2);P;(t1;t2))** In the interval (t1,t2), the attribute t occurs n1 to n2 times.

Chronicle Scenario example

```
chronicle shellcode_mitigation[?source, ?target]{

      event(alarm[ftp_retr_request,?source,?target], t1)
      event(alarm[shellcode,?source,?target], t2)
      noevent(alarm[ftp_transfer_complete,?target,?source], (t1+1,t3-1))
      event(alarm[ftp_transfer_complete,?target,?source], t3)

      t1 < t2 < t3

      when recognized {
         emit event(alarm[shellcode_mitigation, ?source, ?target], t2)
      }
}
```

Chronicle - Discussion

- Expressiveness: Compact way for describing attacks, Different functions that enrich the expressivity time-constraints, non-occurence, ...
- Unambiguous.
- Online/Offline: Online.
- Simplicity: Simple way to describe complex attacks.

Expert Systems: Lambda

- Part of exploit systems.
- Describes the attack from the attacker point of view.
- Each attack is divided into the following:
  - pre-condition.
  - scenario.
  - post-condition.

F. Cuppens and R. Ortalo.
"Lambda: A language to model a database for detection of attacks".
Proceedings of the Third International Workshop on Recent Advances in Intrusion Detection, Springer-Verlag, pp. 197–216, 2000.

F. Sadri and R. A. Kowalski.
"Variants of the event calculus".
International Conference on Logic Programming, pp. 67–81, 1995.

Scenario Example

Action *touch(Agent,File)* *Pre: true* *Post: file(File), owner(Agent,File)*	**Action** *block(Agent,Printer)* *Pre: printer(Printer),* *physical_access(Agent,Printer)* *Post: blocked(Printer)*
Action *lpr-s(Agent,Printer,File)* *Pre: printer(Printer), file(File),* *authorized(Agent,read,File)* *Post: queued(File,Printer)*	**Action** *remove(Agent,File)* *Pre: owner(Agent, File)* *Post:* not*(file(File))*
Action *ln-s(Agent,Link,File)* *Pre:* not*(file(Link))* *Post: linked(Link,File)*	**Action** *unblock(Agent,Printer)* *Pre: printer(Printer), blocked(Printer),* *physical_access(Agent,Printer)* *Post:* not*(blocked(Printer))*
Action *print-process(Printer,Link)* *Pre: queued(Link,Printer),* *linked(Link,File),* not*(blocked(Printer))* *Post: printed(Printer,File),* not*(queued(Link,Printer))*	**Action** *get-file(Agent,File)* *Pre: printed(Printer,File),* *physical_access(Agent,Printer)* *Post: read_access(Agent,File)*

Discussion

- Interesting in the trace analysis
- Accumulation, inference and decision making is useful to detect maybe unkown attacks.
- Interesting way of dealing with synthetic events (Knoweldge database).
- Simple way of describing attacks (pre, scenario and post conditions).
- Describing the attacks from the attacker point of view.

Agenda

Name	1	2	3	4	5	6	7	8	9
Snort	-	-	-	-	✓	✓	✓	-	-
SECnology	-	-	✓	-	✓	✓	✓	-	-
Blare	-	✓	-	-	-	✓	✓	✓	✓
BlueBox	-	✓	-	-	-	✓	✓	✓	✓
RUSSEL	✓	✓	✓	✓	-	✓	-	-	-
BRO	✓	✓	✓	✓	-	✓	-	-	-
DTrace	✓	✓	✓	✓	✓	-	-	-	-
SystemTap	✓	✓	✓	✓	✓	-	-	-	-
STATL	✓	✓	✓	✓	-	-	✓	✓	-
SMC	✓	✓	✓	✓	✓	-	-	-	-
Ragel	✓	✓	✓	✓	✓	-	-	-	-
BSML	✓	✓	✓	✓	✓	-	-	-	-
IDIOT	✓	✓	✓	✓	✓	-	-	-	-
ADele	✓	✓	✓	✓	✓	-	-	-	-
Chronicle	✓	✓	✓	✓	✓	✓	-	✓	-
LogWeaver	✓	✓	✓	✓	✓	-	-	-	-
P-Best	✓	✓	✓	✓	✓	-	-	-	✓
Lambda	✓	✓	✓	-	✓	-	✓	-	✓

1. Sequence of event.
2. Non-occurrence of events.
3. Time constraint.
4. Number of occurrence of an event
5. Context sensitive.
6. Online analysis.
7. Simplicity.
8. Suitable for kernel tracing.
9. Possibility of inferring new facts.

Agenda

Conclusion

Scenario-based

Unsuitable for detecting unknown malicious behaviors.

Policy-based

Too restricted and could generate a lot of false alarms.

Properties

There is no language that covers all the properties studied so far.

Conclusion

Scenario-based

Unsuitable for detecting unknown malicious behaviors.

Policy-based

Too restricted and could generate a lot of false alarms.

Properties

There is no language that covers all the properties studied so far.

Conclusion

Scenario-based

Unsuitable for detecting unknown malicious behaviors.

Policy-based

Too restricted and could generate a lot of false alarms.

Properties

There is no language that covers all the properties studied so far.

Future Work

The target is to define a language that is both scenario and policy based:

scenario-based

To prevent not only known attacks but also unknown variations of attacks or attacks that exploit similar mechanisms.

policy-based

To prevent abnormal behaviors given a set of policy rules rather than models that specify "normal" behaviors.

Future Work

Milestone K1.3

Implement an automated fault pattern detection engine, based on the selected pattern description language. Describe a number of low level problems using the selected language.

Milestone K1.4

Measure the performance of the fault pattern detection engine on large traces when simultaneously searching for several patterns. Measure the accuracy of the patterns for detecting the problematic conditions. Adapt and optimize the algorithms.

Milestone K1.5

Publish the new fault patterns descriptions, description language and fault pattern detection algorithms developed.

Relations with other teams

- **Trace Abstraction project**: The scenario is based on abstracted events to be more generic and cover similar attacks.
- **System Health project**: The results of the detection could be an input to the system health to calculate the safety and the performance of the system.

Expected Results (for the short term)

- An Eclipse plug-in editor for the definition of scenarios and policy rules.
- An Eclipse plug-in engine that automatically detect faults in an abstracted version of LTTng traces.

Thank you

Questions?!