# Hardening Registration Number Protection Schemes against Reverse Code Engineering with Multithreaded Petri Nets

Talk at RECON2005

Thorsten Schneider

# Overview

- Introduction
- Petri Nets – Overview
- Petri Nets – Example
- Protection by Obscurity and Obfuscation
- Example: Protection with Petri Nets
- Discussion
- Results

# Known Problems

**Software Piracy**
- Crackz
- Keygenz

**Plagiarism**
- Code Ripping
- Algorithm Theft

**Viruses**
- Encryption
- Hiding

Problem

**Reverse Code Engineering**
- Disassembly (Static + Dynamic)
- Decompilation
- Code Transformation
- Code Flow Reconstruction
- Analysis + Understanding

# One Method of Resolution

**Decrease Code Understanding**

- Hardening through Obscurity
- Hardening through Obfuscation
- Hardening through Complexity
- Manipulation of Code Flow Graphs
- Manipulation of Information Flow Graphs
- …
- Petri Nets!

```
➢From: "Jim Coplien" <cope@research.bell-labs.com>
> Date: Tue, 22 Dec 1998 13:03:56 -0600
> To: Lalita Jagadeesan <lalita@research.bell-labs.com>, god, tball
> Subject: a program for your flow and testing tools
>
> /*
> * seriously -- run it :-)
> */
> #include <stdio.h>
> main(t,_,a)
> char *a;
> {
> return!0<t?t<3?main(-79,-13,a+main(-87,1-_,main(-86,0,a+1)+a)):
> 1,t<_?main(t+1,_,a):3,main(-94,-27+t,a)&&t==2?_<13?
> main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-72?main(_,t,
>"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#{l+,/n{n+,/+#n+,/#\
> ;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
> q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw' i;#\
> ){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
> iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
> ;;{nl'-{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
> }'+}##(!!/")
> :t<-50?_==*a?putchar(31[a]):main(-65,_,a+1):main((*a=='/')+t,_,a+1)
> :0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a,
> "!ek;dc i@bK'(q)-[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);
> }
```

# Petri Nets: Overview

- A *Petri Net* is a method, to represent *processes* in an abstract way
- Uninteresting Processes (for us):
  - Factory work flows
  - Business flows
  - Communication flows (protocols)
  - Device controls
  - Handicraft manuals
  - Biological Pathways (Bioinformatics)
- Interesting Processes:
  - Software Development Processes
  - Software Algorithms
  - Registration Number Schemes
- Petri Nets are graphs
- Advantage: Multithreaded processing!

# Petri Net Types

- Discrete Petri Net
- Autonomous Petri Net
- Non-Autonomous T-Timed and P-Timed Petri Net
- Stochastic Petri Net
- Continuous Petri Net
- CSPN (Constant Speed Petri Net)
- VSPN (Variable Speed Petri Net)
- Hybrid Petri-Net

# Petri Nets: Formal Definition

A Petri Net is a 6-Tupel $(S,T,F,K,W,M_0)$ with:

- S: non-empty set of locations (Places)
- T: non-empty set of Transitions
- F: non-empty set of edges (Arcs)

---

- K: Capacity of Places for Tokens
- W: Weight of Edges
- $M_0$: Startup Marking

$S=\{s_1, s_2, \dots, s_{|S|}\}$
$T=\{t_1, t_2, \dots, t_{|T|}\}$
$F \subseteq (S \times T \cup T \times S)$
$K: S \to N \setminus \{0\}$
$W: F \to N \setminus \{0\}$
$M_0: S \to N$

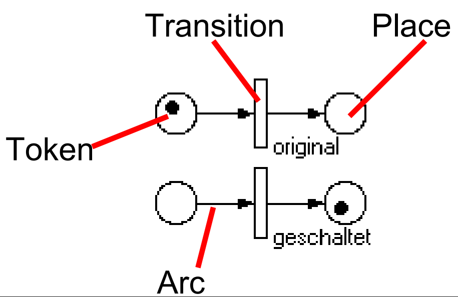
$0 \le M(s) \le K(s)$
$S \cap T = \emptyset$

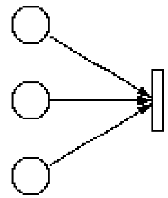
**Very Simplified:**

- 3 different objects: Places, Transitions and Tokens
- No object (Places, Transitions) can belong to both sets
- Between Places and Arcs there *might* be a relation (F)
- Can simulate „something“

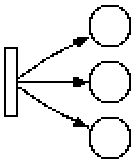
- So a Petri Net is a kind of a runable process graph

# Petri Nets: Basics

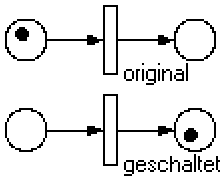


Before Transition



After Transition




Firing Process

- Firing of *Transitions* changes network *Tokens* (located at the Places)
- Only one Transition can fire
- If so, a Transition removes as many Tokens, as the *Weight* of the *Arcs* defines
- The Places after the Transition receive the Tokens

- The Places *before* the Transition need to have enough Tokens
- The Places *after* the Transition need to have enough empty space for new Tokens.

- A Transition which is *able* to fire is called *activated*
    - But: activated does *not* mean that it is really fired!

- A Petri Net containing no activated Transitions is a *dead* Petri Net

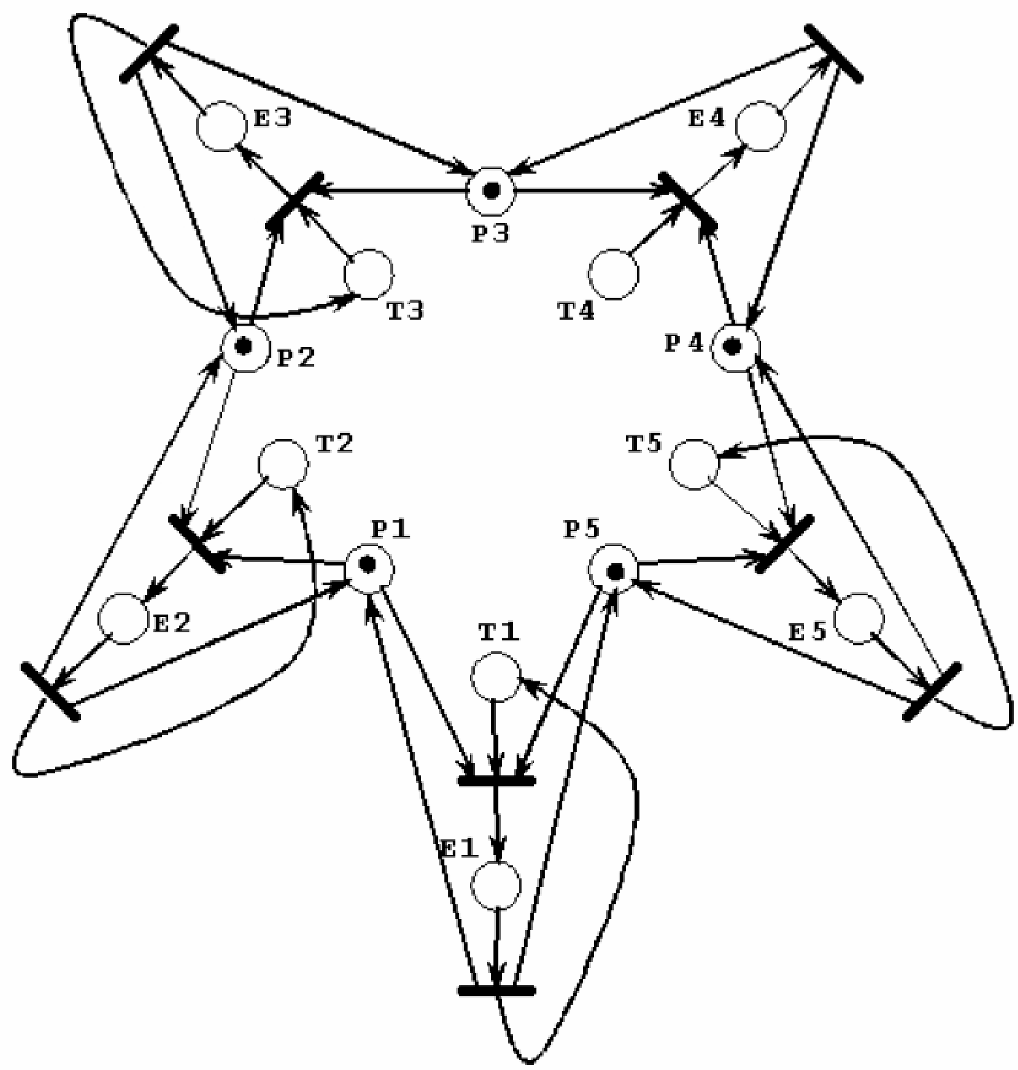
# Petri Nets might save lifes!

**The Five Chinese Sages Problem [Dijkstra]:**

*Five Chinese sages are sitting at the circle table and have a dinner. Between of each two sages is only one stick. But for eating each of them needs two sticks in a moment. Obviously, if all sages takes sticks from the left side and waiting sticks from right side they all will die through starvation (dead loop).*

DIJKSTRA, E.W.: *Co-operating sequential processes.* In Programming Languages, F. Genuys, Ed., 43–112, 1968.

# Petri Nets might save lifes!




- Places $P_1 ... P_5$ introduce sticks
- All sticks are on the table at the first moment
  - → each place has a marker inside
- Transitions $T_i$ and $E_i$ introduce sages states:
  - $T_i \rightarrow sage_i$ thinks
  - $E_i \rightarrow sage_i$ eats.
- To pass from $M_i$ state (obviously, no one can satisfy his hunger through his thoughts) to $E_i$ state, both sticks (on left and right sides) must be on the table at one moment.

# Petri Nets: Conflicts

- **Pre-Conflict:**
  - 2 Transitions need the *same* Token to Fire
  - Both Transitions are *activated,* but *only 1* can fire
  - This is no erroneous Petri Net, but models the decision between 2 alternatives

- **Post-Conflict:**
  - Similar to Pre-Conflict
  - 2 Transitions produce Tokens, but the capacity of the Places is to low for all Tokens
  - Solution is dependant on conflict strategy

- **Confusion:**
  - Is a doubled conflict
  - One Transition conflicts with two different Transitions

# Protection by Obscurity and Obfuscation

P = Program, T = Transformation, S = Source Code

**Given a program P and an obfuscated program P‘:**

- P‘ has the same observable behavior as P, i.e., the transformations are semantics-preserving.
- The obscurity of P‘ maximized, i.e., understanding and reverse engineering P‘ will be strictly more time-consuming than understanding and reverse engineering P.
- The resilience of each transformation $T_i(S_j)$ is maximized, i.e., it will be difficult to construct an automatic tool to undo the transformations
- The stealth of each transformation $T_i(S_j)$. is maximized, i.e., the statistical properties of $S'_j$ are similar to those of $S_j$.
- The cost (the execution time/space penalty incurred by the transformations) of P‘ is minimized.

Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection, Christian S. Collberg and Clark Thomborson

# Protection by Obscurity and Obfuscation

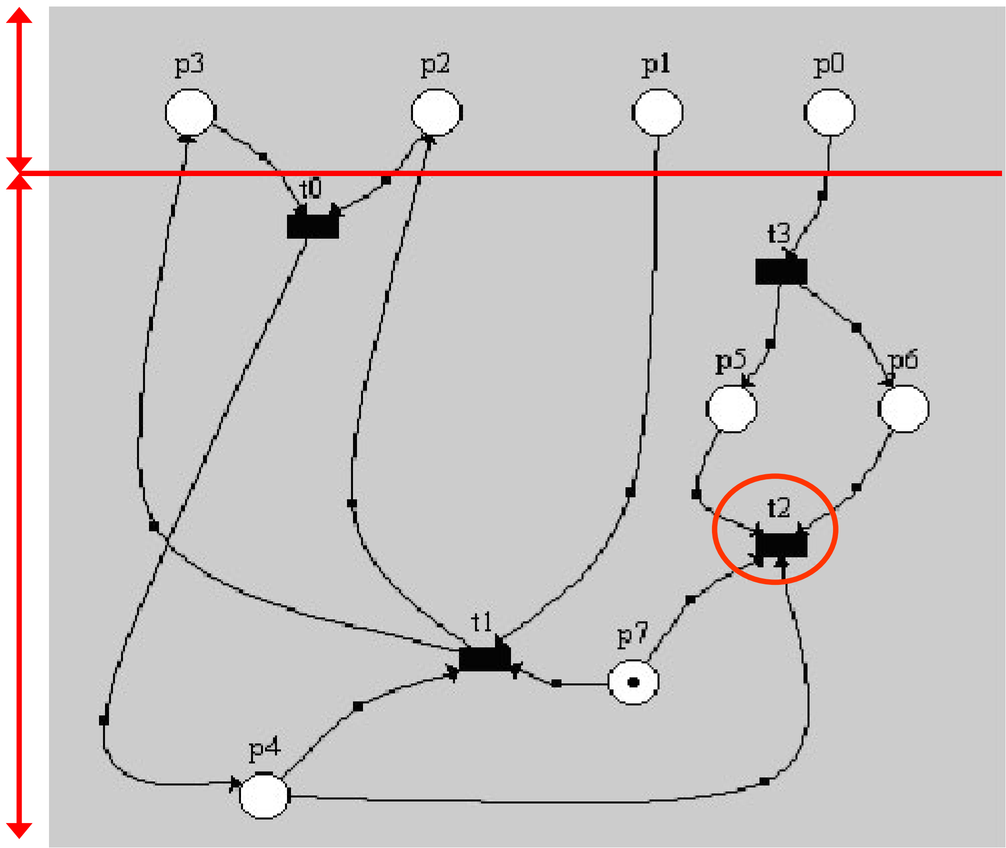
Code obfuscation is very similar to code optimization, except:

- with obfuscation, we are maximizing obscurity while minimizing execution time
- with optimization, we are just minimizing execution time.

Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection, Christian S. Collberg and Clark Thomborson

# Example: Protection with Petri Nets

Registration Number Input




Registration Number Check

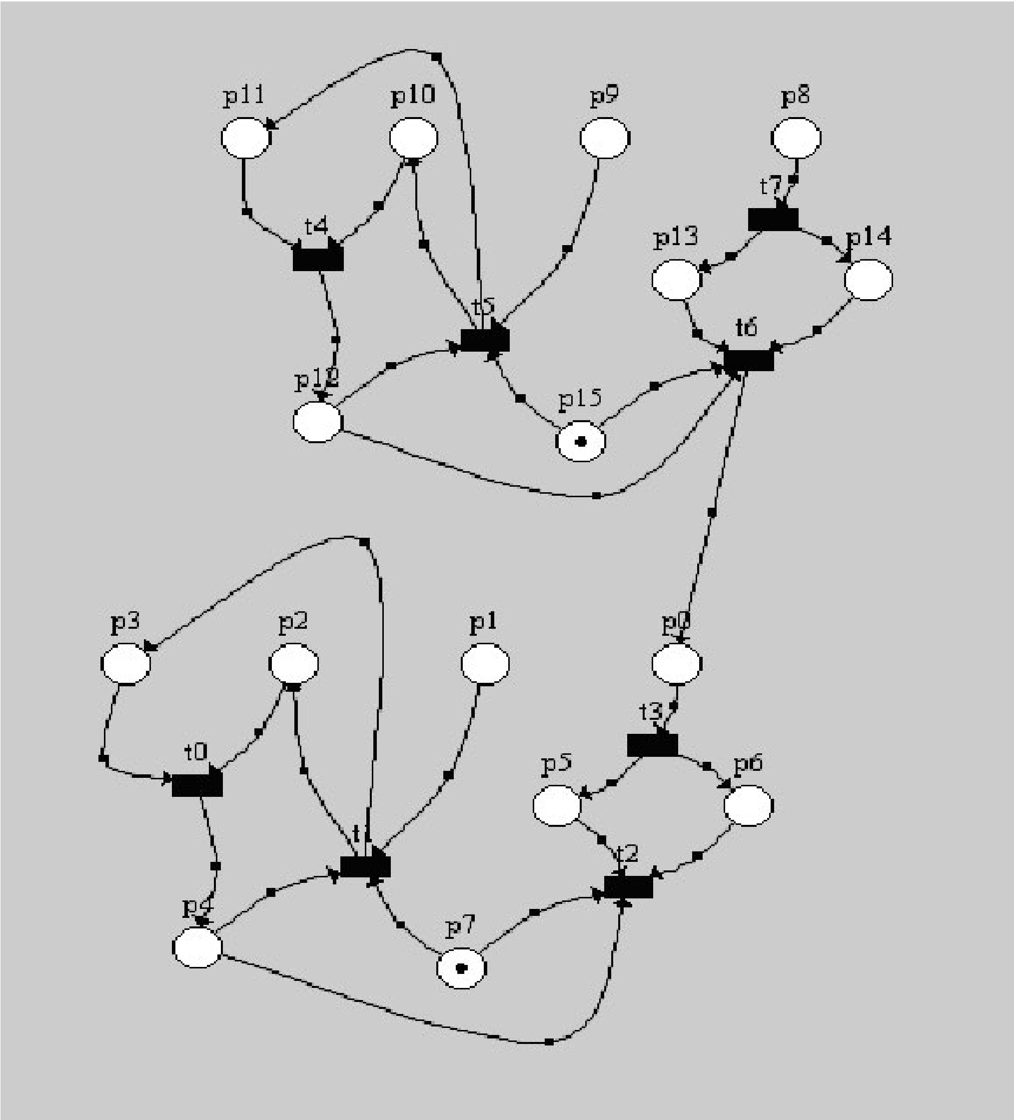
- Markable: $P_0$ to $P_3$
- $P_7$ is pre-marked !
- Lowest Priority to $T_2$
- Application gets registered when $T_2$ is fired

<u>Solution:</u>

$P_0 = 1$
$P_1 = 0$
$P_2 = 1$
$P_3 = 1$

Solution Key Space: $2^4 = 16$ tries
→ Simple for Bruteforce
→ Simple for Brain

# Example: Protection with Petri Nets




- Increasing Complexity
  → Decreasing Understanding
- Simple Copy & Paste possible
  → $P_1$ not possible
  → Reachability Problem of $T_2$
- Introduction of new Places and Transitions might be necessary

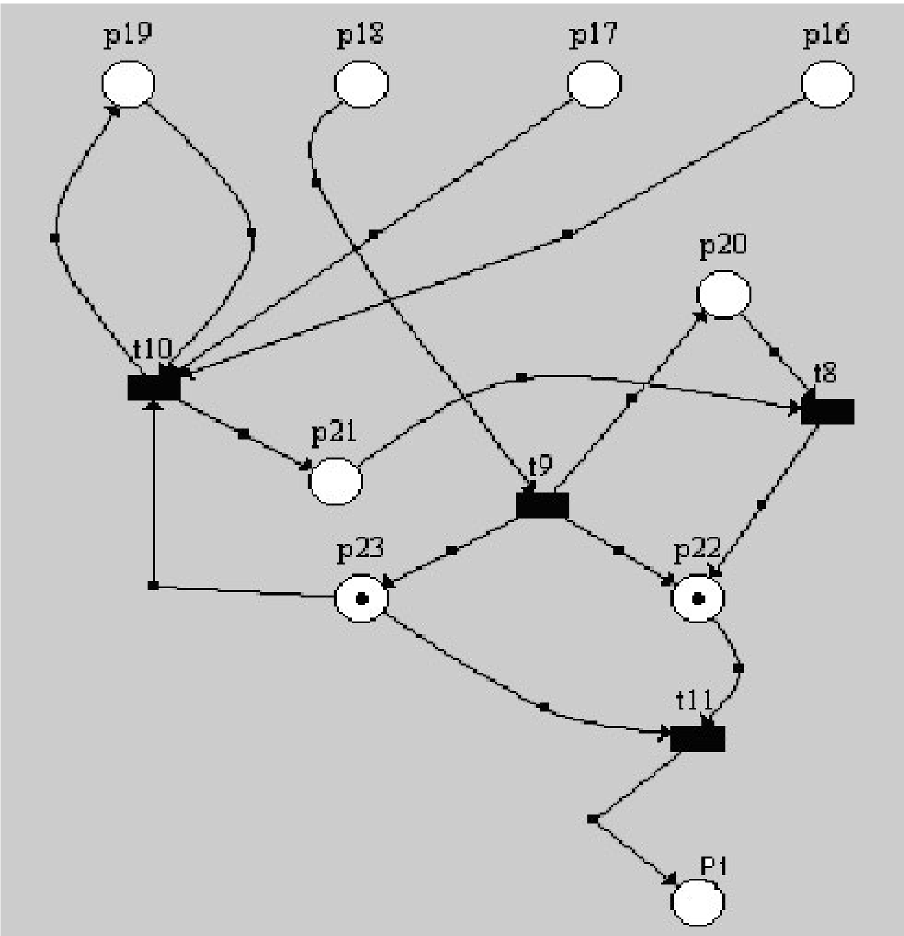
<u>Solution:</u>

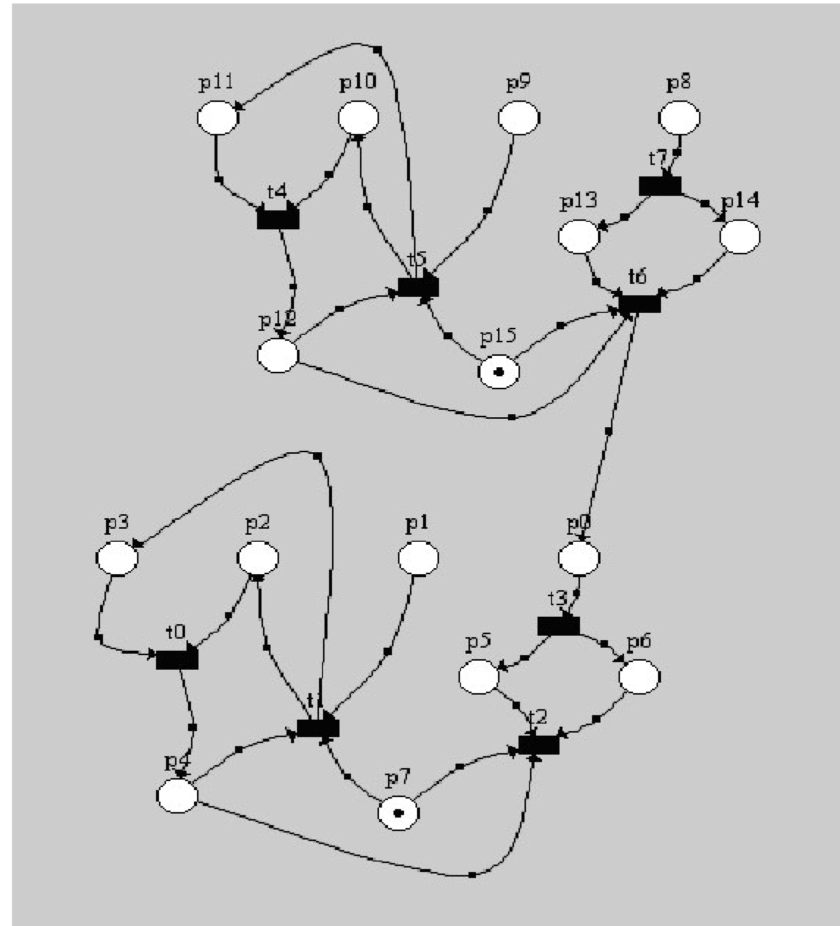
$P_1 = 0$, $P_2 = 1$, $P_3 = 1$, $P_8 = 1$,
$P_9 = 0$, $P_{10} = 1$, $P_{11} = 1$

Solution Key Space:
$2^{16} = 65536$ tries
→ Harder for Bruteforce
→ Harder for Brain

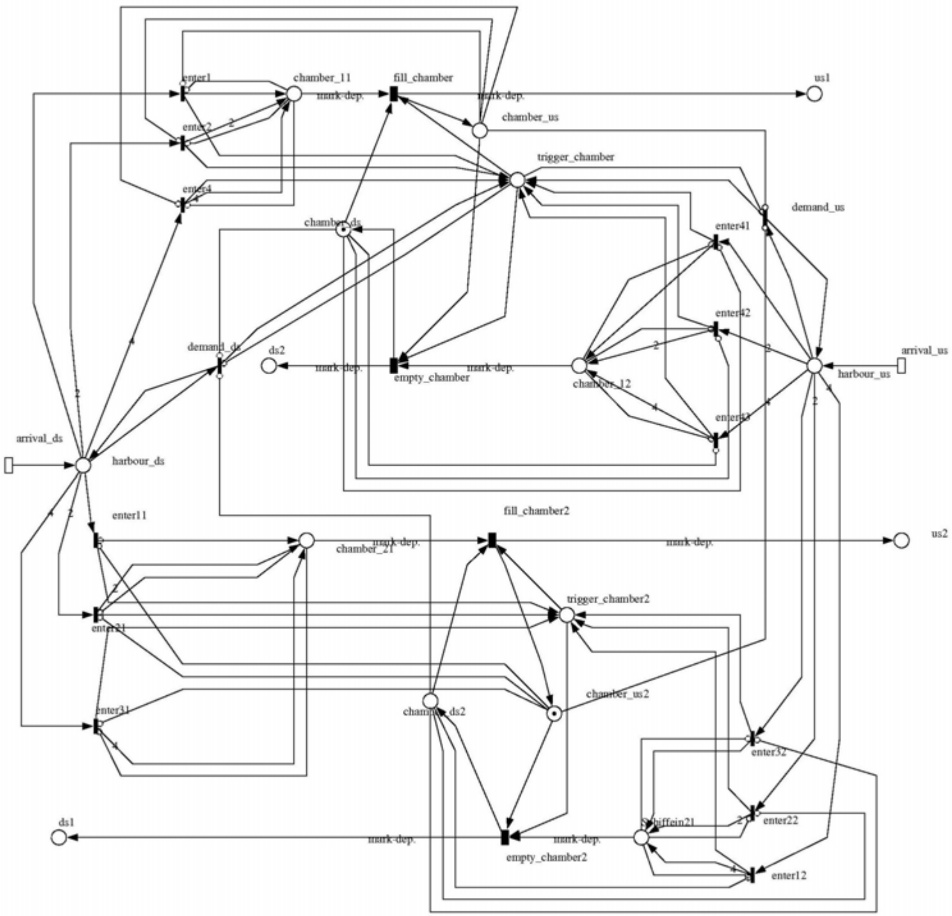
# Example: Protection with Petri Nets

Changed Sub Petri Net to attach at Place  at Place $P_1$



# Example: Protection with Petri Nets




**<u>Now imagine:</u>**

- Each Transition is a Thread
- Each Place is a Thread

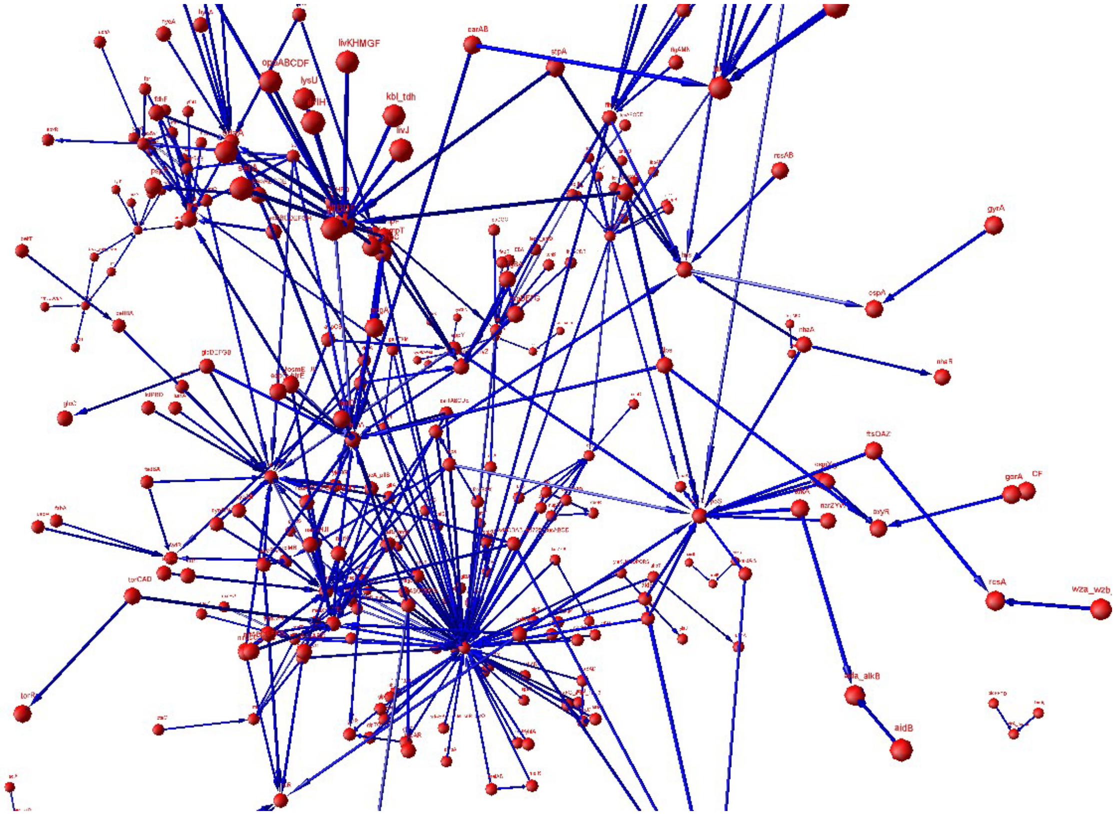
- Each Thread is protected different:
  - Anti-Debugging
  - Anti-Disassembly
  - Self-Encryption
  - Watchdogs
  - ... and much more …

- Each Arc is an encrypted communication protocol

- Each Token is an encrypted Object

***How would you analyse this ?***

# Example: Protection with Petri Nets




**<u>Now imagine:</u>**

1. You have a disassembly
2. You try to reconstruct the Petri Net from disassembly or debugging
3. You need to trace or debug *parallel* processes to understand the parallel processes
4. And all these ugly protection tricks within each Thread!

***How would you analyse this ?***

# Problems of Complexity

- Research only focus on *decreasing* complexity
  → Many research groups
  → Much research

- We want to *increase* complexity
  → No research groups
  → No research yet
  → But (!): decreasing complexity can be inverted!
  → But: No algorithms yet

# Discussion: Pro

- High complexity
- High obscurity
- Reconstruction of Petri Net from binary code is hard up to impossible
- Protection is hard up to impossible to understand

# Discussion: Contra

- Once reconstructed, it is possible to simplify the Petri Net
- Once simplified, it is possible to run reduced Bruteforce Attacks
- Once bruteforced, it is possible to get a valid key or Keygen
- Protection still breakable (e.g. Patching) at the Input Layer of the protection
- Development of complex Petri Nets is very time consuming, no automatism yet
- Implementation very time-consuming, no automatism yet

# Results

- Petri Nets are an efficient way to obscure and to complex processes
- Resistant against Bruteforcing
- But: Once analysed, they can be simplified
- Example source and binary available
- Fact: all software protection schemes have been cracked
- Fact: If a code is runable, you can crack it!
- Further research necessary!

# Example Code with Online Disassembly

(http://pvdasm.reverse-engineering.net/PVPHP.php)

# Acknowledgments

- Robert Airapetyan (Polytechnical University of Odessa)
- RECON 2005 Team
- The anonymous reviewers
- The audience

# Advertisement

knoppix|RE

**http://knoppix-re.reverse-engineering.net**

# Questions ?

**Questions ?**