# Program Analysis

## Lecture 13: *Current Research Topics*
## Winter term 2011/2012

Prof. Thorsten Holz

# Anouncements

- Today's lecture ends at ~10am
  - Last exercise afterwards
- Exams on February 22
  - Prepare the exercises you had to solve during the semester and you should pass the exam
  - Bonus points will be announced via Moodle

# Static vs. Dynamic Analysis I

- Static analysis
  - Code is not executed
  - All possible branches can be examined (in theory)
  - Requires lots of expert knowledge
- Problems of static analysis
  - Undecidable in general, approximations necessary
  - Disassembly difficult: obfuscated code + packers

# Static vs. Dynamic Analysis II

- Dynamic analysis (*Anubis*, *CWSandbox*, *BitBlaze*)
    - Code is executed
    - Observe instructions that are actually executed
- Problems of dynamic analysis
    - Only a single path is examined (*multi-path execution*)
    - Analysis environment possibly not *invisible* (NDSS'10)
    - Analysis environment possibly not *comprehensive*

# Motivation

- It might be useful to have full control over the analysis environment
- Virtualization is an interesting technique that might be helpful
  - Program to be analyzed is executed in guest system
  - Our instrumentation is performed in host system
  - We have full control, we can even stop guest and perform arbitrary analysis

# Anubis Overview

# Anubis Overview

- We execute binary sample

  ➡ dynamic analysis

# Anubis Overview

- We execute binary sample
  - ➡ dynamic analysis
- ... within an emulated environment ...
  - ➡ Emulation of a complete PC (CPU + hardware)
  - ➡ Qemu as emulator, uses Windows XP SP2

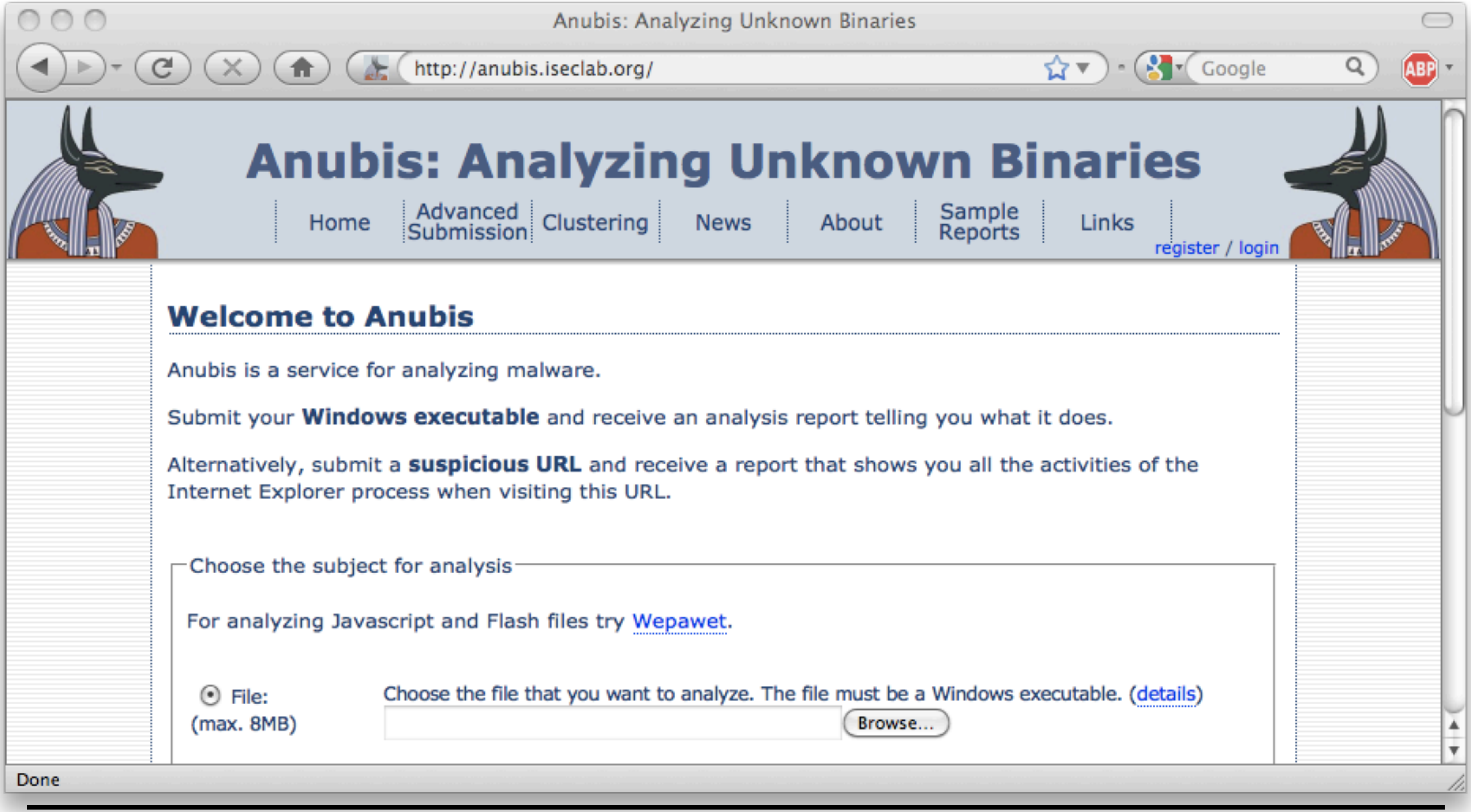
# Anubis Overview

- We execute binary sample
  - ➡ dynamic analysis
- ... within an emulated environment ...
  - ➡ Emulation of a complete PC (CPU + hardware)
  - ➡ Qemu as emulator, uses Windows XP SP2
- ... and observe all activity
  - ➡ *System calls* and *Windows API calls*

# Web Interface

- http://anubis.iseclab.org
  - Enables **an**alysis of **u**nknown **bi**narie**s**
  - Results via web browser or e-mail
- Analyzed several million binary samples
- Used by people around the world
- Exchange of information with AV vendors and security companies

# Web Interface



## Anubis: Analyzing Unknown Binaries

### Welcome to Anubis

Anubis is a service for analyzing malware.

Submit your **Windows executable** and receive an analysis report telling you what it does.

Alternatively, submit a **suspicious URL** and receive a report that shows you all the activities of the Internet Explorer process when visiting this URL.

Choose the subject for analysis

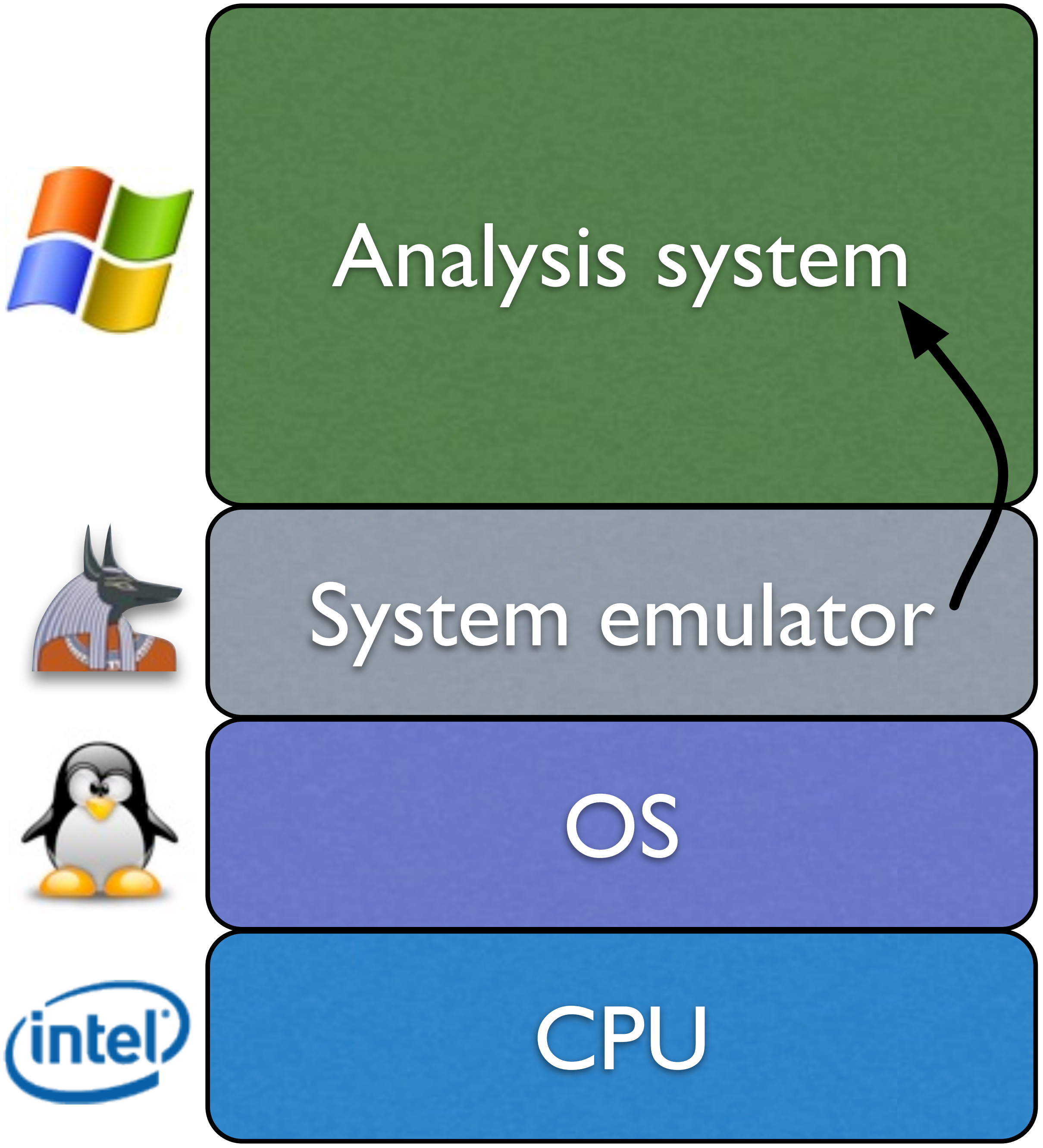
For analyzing Javascript and Flash files try Wepawet.

○ File: (max. 8MB) Choose the file that you want to analyze. The file must be a Windows executable. (details)

# Web Interface

- http://anubis.iseclab.org
  - Enables **an**alysis of **u**nknown **bi**narie**s**
  - Results via web browser or e-mail
- Analyzed several million binary samples
- Used by people around the world
- Exchange of information with AV vendors and security companies

# Intuition



- *Virtual Machine Introspection*
- Programm analysis techniques
  - Taint analysis
  - Symbolic execution
  - *Program Slicing*
  - *Analysis of information flow*

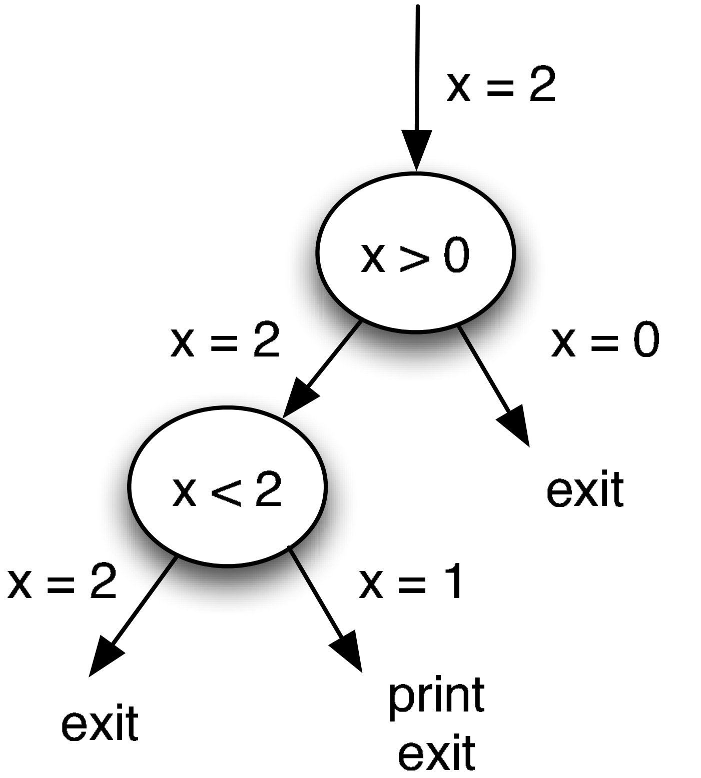
# Multi-Path Execution

```
int x;
x = read_input();
if (x > 0)
    if (x < 2)
        printf("ok");
exit(0);
```

- We want to traverse all paths, find all conditions
- Iteratively explore the different paths
  - Store execution state and reset state if necessary

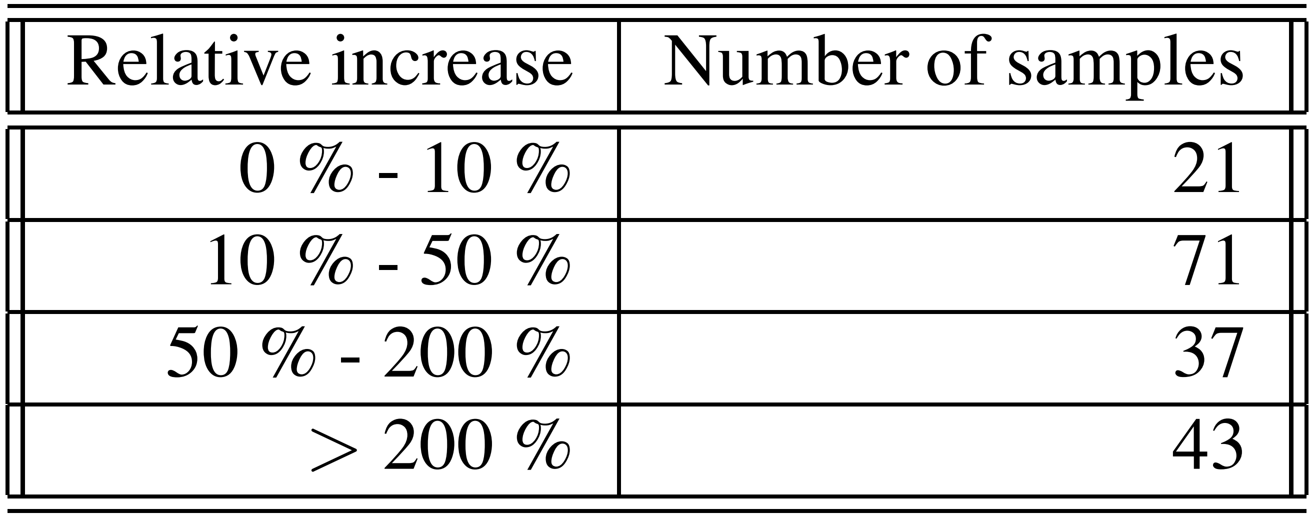
# Multi-Path Execution

```
int x;
x = read_input();
if (x > 0)
    if (x < 2)
        printf("ok");
exit(0);
```



- We want to traverse all paths, find all conditions
- Iteratively explore the different paths
    - Store execution state and reset state if necessary

# Multi-Path Execution



| Relative increase | Number of samples |
|---:|---:|
| 0 % - 10 % | 21 |
| 10 % - 50 % | 71 |
| 50 % - 200 % | 37 |
| > 200 % | 43 |

- Works with real-world malware samples
- Limitations
  - State explosion, especially for memory
  - Network state is lost on restore

# Inspector Gadget

# Inspector Gadget

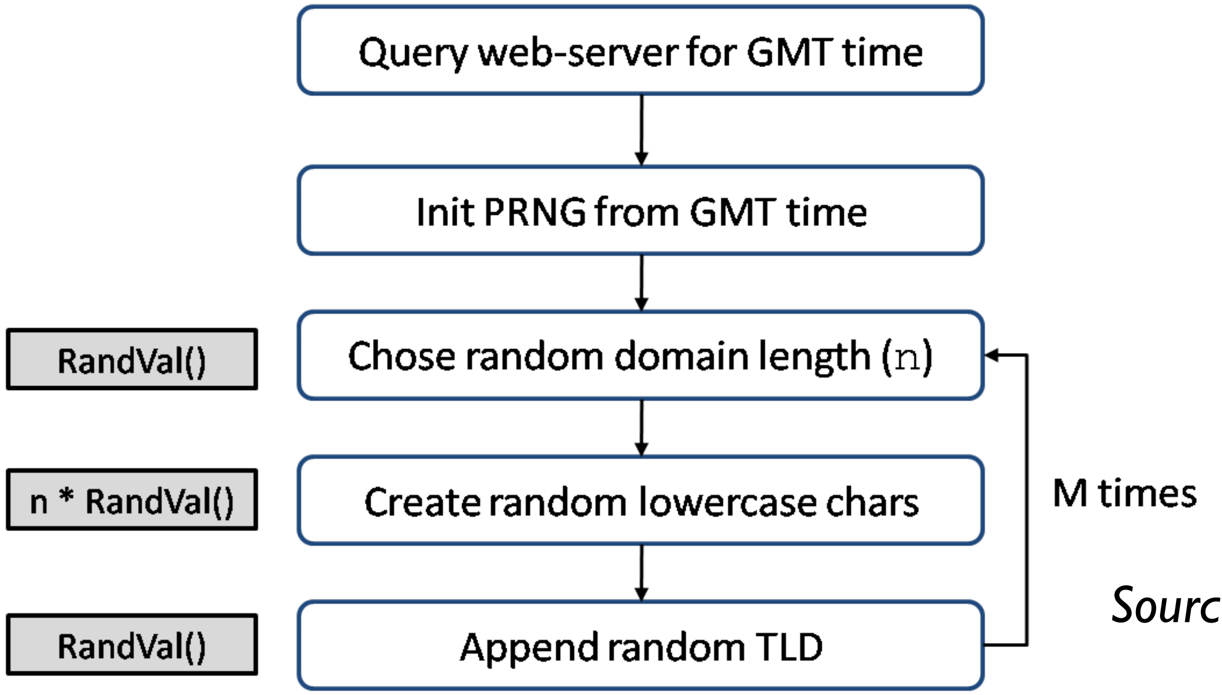
- Malware analyst is typically interested in specific *algorithm* implemented by malware sample
  - Domain generation algorithm of Conficker
  - Binary update for bots
  - Template download of spambots
- Can we automatically extract the algorithm?

# *Conficker*

- *Conficker* algorithm to compute domains
  - *Which C&C domains will be used?*
  - Depends on current date, retrieved by issuing an HTTP request to external webserver

# Conficker

- *Conficker* algorithm to compute domains
  - *Which C&C domains will be used?*
  - Depends on current date, retrieved by issuing an HTTP request to external webserver



Query web-server for GMT time

Init PRNG from GMT time

RandVal() — Chose random domain length (n)

n * RandVal() — Create random lowercase chars

RandVal() — Append random TLD

M times

*Source: "KYE: Containing Conficker"*
*F. Leder, T. Werner*

# Motivation

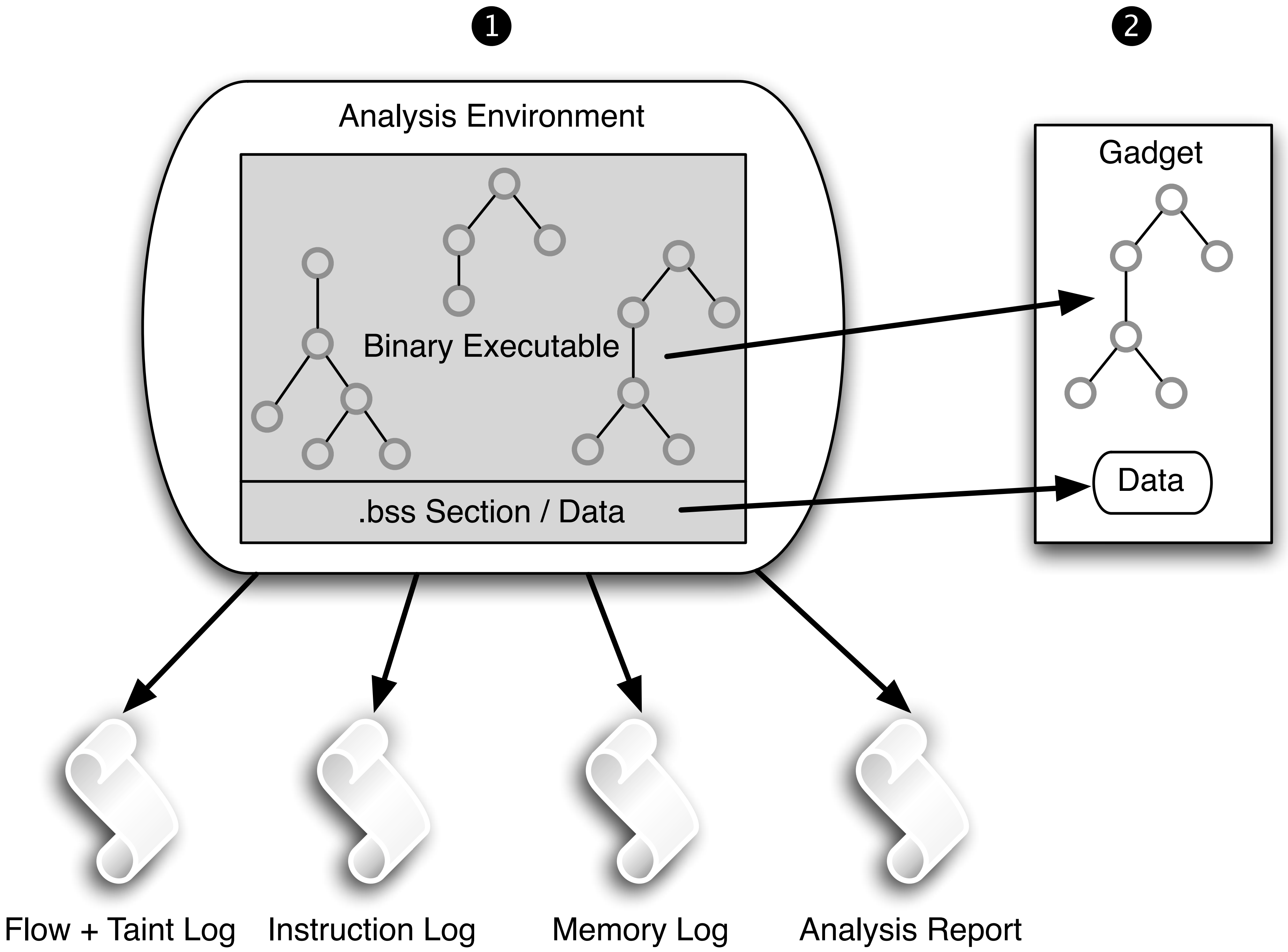
Question:
*Can we extract from a given binary sample a certain behavior in an automated and efficient way?*

*Input:* binary executable

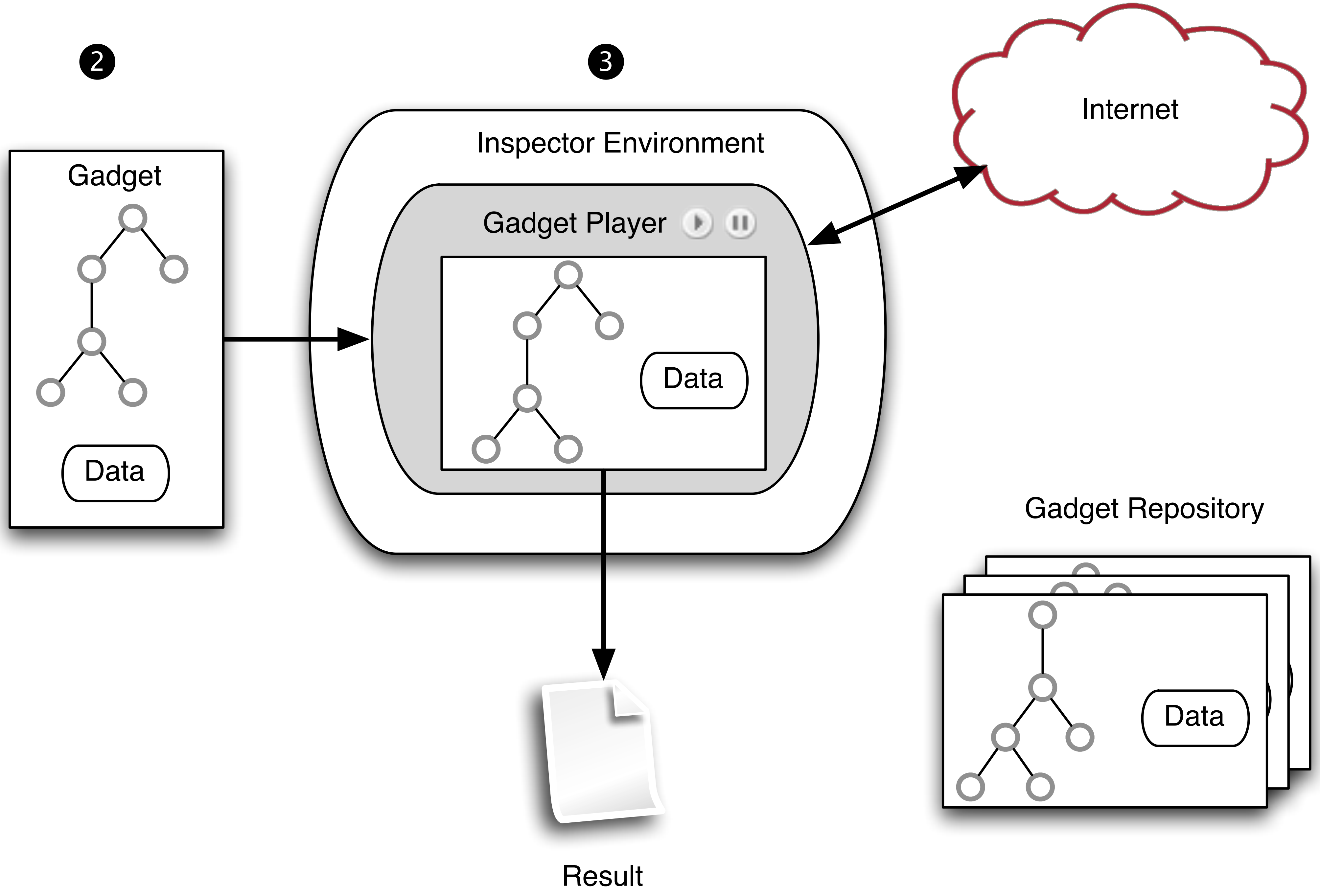
*Output:* extracted algorithm that can be executed in an autonomous way

Kolbitsch et al., IEEE S&P 2010

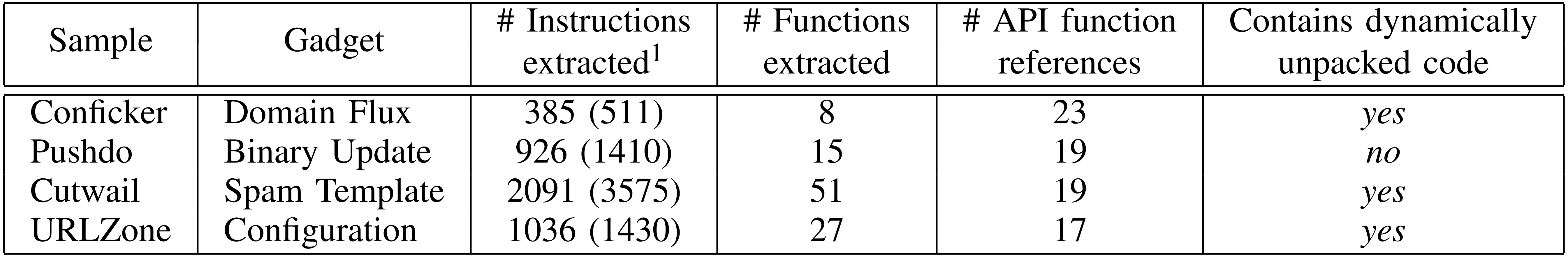
# Overview


❶
❷
Analysis Environment
Binary Executable
.bss Section / Data
Gadget
Data
Flow + Taint Log
Instruction Log
Memory Log
Analysis Report

# Overview


❷
❸
Internet
Inspector Environment
Gadget
Gadget Player
Data
Data
Gadget Repository
Data
Result

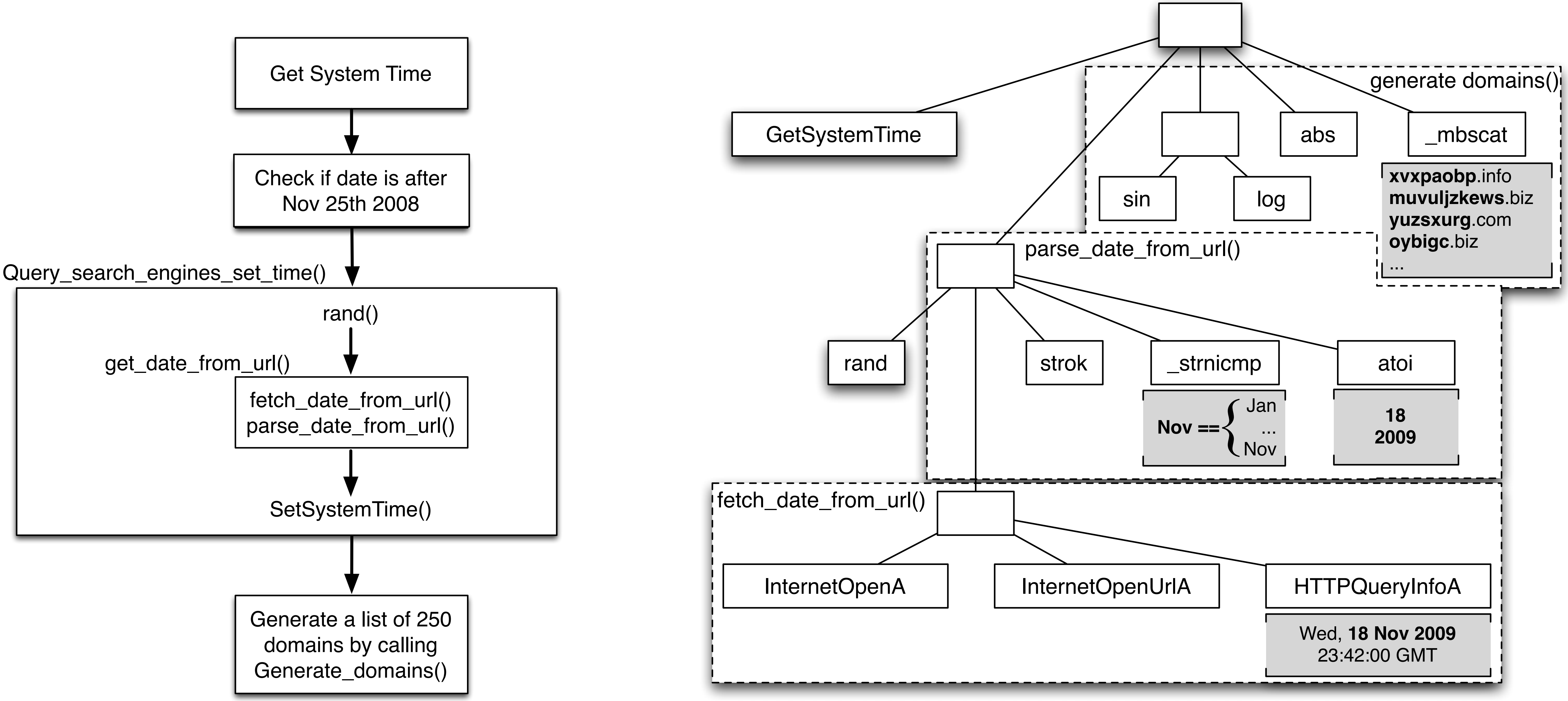
# Evaluation



| Sample | Gadget | # Instructions extracted[1] | # Functions extracted | # API function references | Contains dynamically unpacked code |
|---|---|---|---|---|---|
| Conficker | Domain Flux | 385 (511) | 8 | 23 | *yes* |
| Pushdo | Binary Update | 926 (1410) | 15 | 19 | *no* |
| Cutwail | Spam Template | 2091 (3575) | 51 | 19 | *yes* |
| URLZone | Configuration | 1036 (1430) | 27 | 17 | *yes* |

- Works on real-world malware
- Automated extraction of specific algorithm
  - Analyst needs to specify what she is interested in
  - But we might not include all relevant code

# Conficker

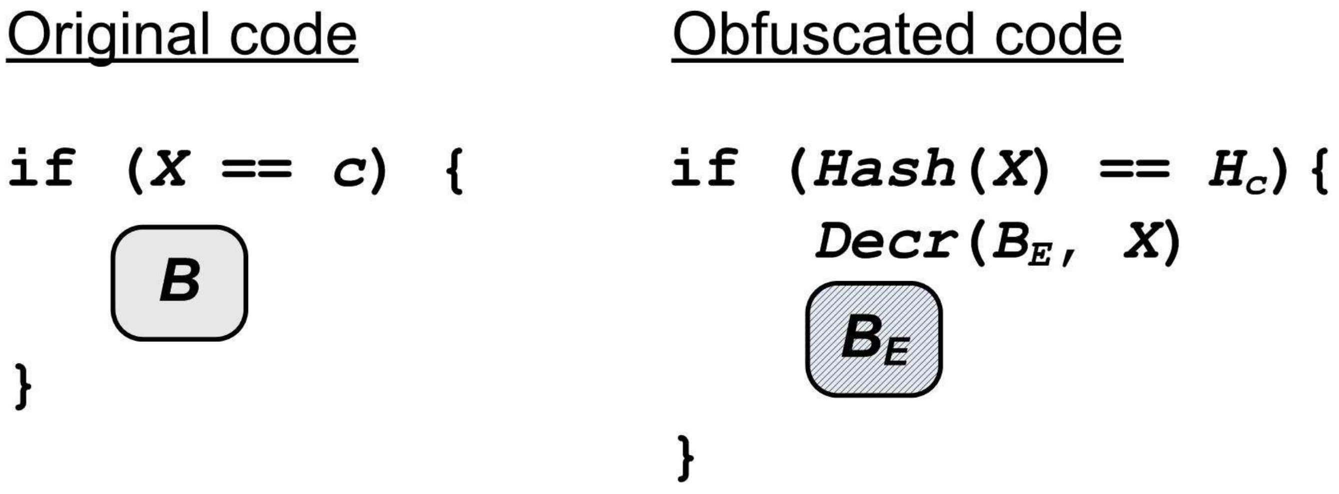


*Source: Porras et al.: "A Foray into Conficker's Logic and Rendezvous Points", LEET'09*

# Conficker

```
ysodtmnq.org
ylkwa.org
fvgwbijrih.com
byhmvpxynvn.biz
lxggsrccct.info
igvnycyx.info
kdathezjp.com
rqeubkg.net
fcmremoo.org
ilcsea.com
eydxnhtqpoj.net
pocomecwapn.biz
aoxcuetqqm.com
jgxlrqkc.biz
yswtxkeo.biz
hcmem.biz
dcinu.net
fxgoswinvuc.com
mnxtest.info
gwfvi.org
btrgonoq.org
emohwqe.org
sppmfy.org
zhwpjb.org
iqgvjuaz.com
bnhvv.com
dicntiqhv.info
cndbwljc.info
xqyeoncaooq.info
pplkxgcv.biz
yjjry.info
vejfoshm.net
uvnff.info
origmhqf.net
vealjox.com
hsxwcogs.biz
fjvddjusifj.org
lgpdk.net
tuqeszsnx.org
sflxwx.com
yqecdzyf.net
jyoacdcj.biz
cttqmf.com
iysfbrspiam.com
pscrdwugwa.com
yxvtwzw.com
bujykwamdgf.info
tddhmrbtdo.net
jqwjdig.biz
...
```

# Challenges



Original code

```
if (X == c) {
    B
}
```

Obfuscated code

```
if (Hash(X) == H_c){
     Decr(B_E, X)
     B_E
}
```

Where, $H_c$ = *Hash*(*c*) , $B_E$ = *Encr*( *B*, *c* )

- Conditional code-obfuscation (Sharif et al., NDSS'08)
  - Only observe behavior seen during execution
  - Attacks against MPE, symbolic execution, and forced conditional execution

# Challenges

- Virtual machine based packers
  - “virtual CPU” executes packed code
  - Impedes static and also dynamic analysis
  - Automatically cross-compile to x86?
- Analysis of kernel-based malware
  - Context not always clear
  - Tricky use of OS specific techniques

# Summary

- Different approaches for automated analysis
  - Dynamic approaches help a lot
  - *Inspector Gadget* to automatically extract algorithms
  - Works with real-world malware, but still prototype
- Many open problems remain
  - Especially related to packers, kernel-level malware, ...

# Questions?

Contact:

Prof. Thorsten Holz

thorsten.holz@rub.de
@thorstenholz on Twitter

More information:

http://syssec.rub.de

http://moodle.rub.de