SIDAR

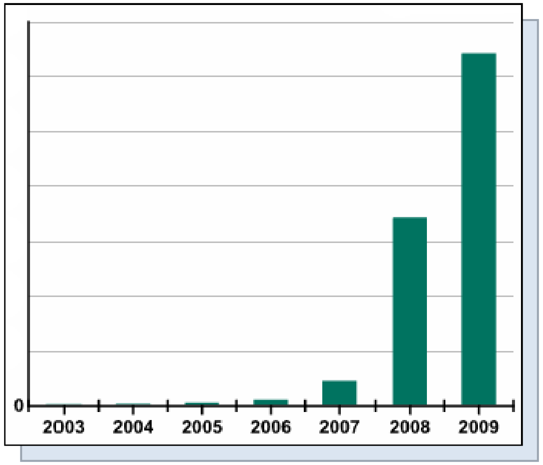
SPRING 5 – 07.07.2010, Bonn

GI Graduate Workshop
on Reactive Security

PYTHON SANDBOX

Daniel Plohmann and Felix Leder
University of Bonn, Germany
{plohmann, leder}@cs.uni-bonn.de

Overview



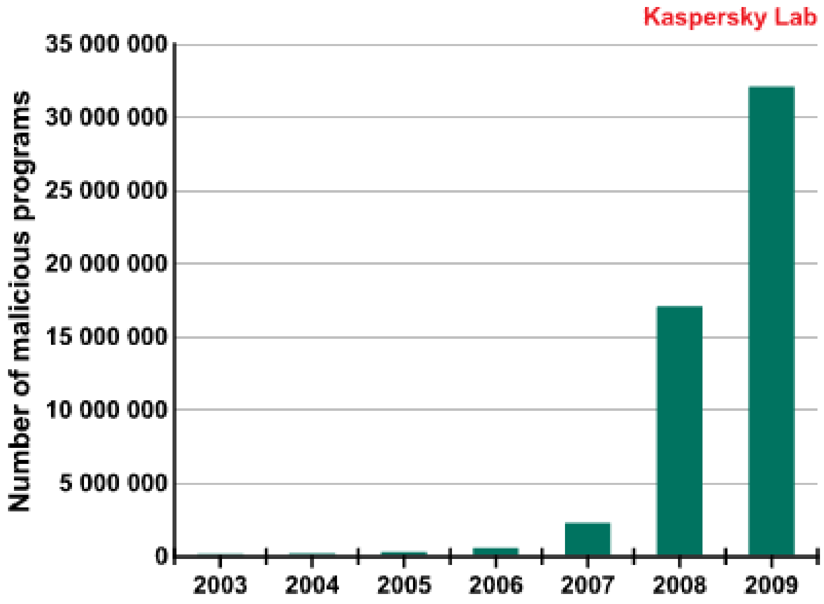
1. Motivation

2. Approach

3. Live Demo

Motivation

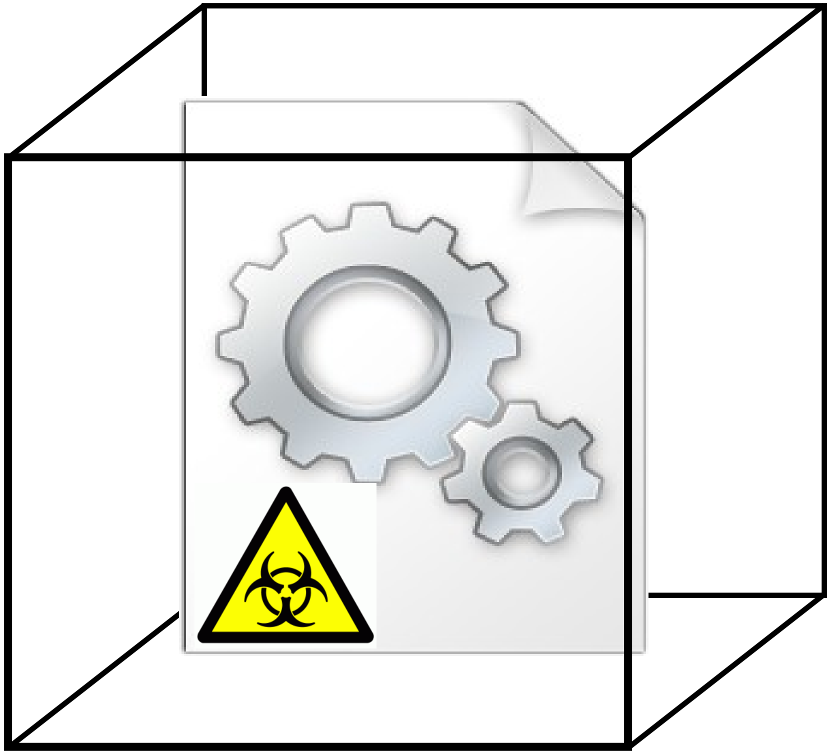
- Annual Reports 2009:
  ~ 55.000 new samples per day (PandaLabs)
  ~ 90.000 unique ZeuS binaries (Symantec)
  2,895,802 new malware signatures (Symantec)



→ Automation needed!

Motivation

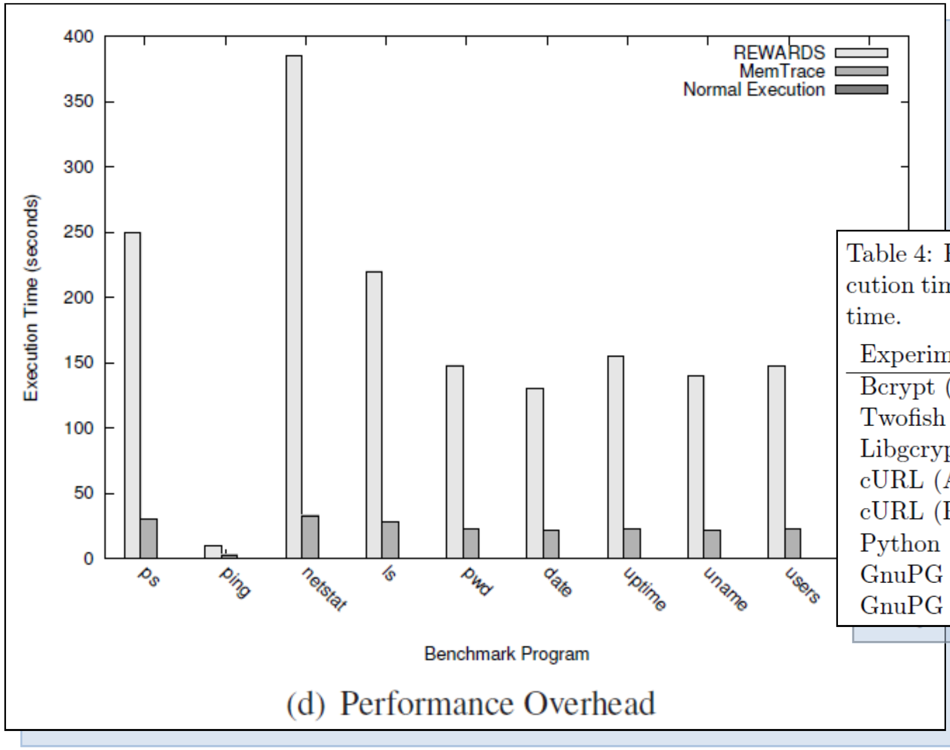
- Common approach to automation: sandboxing



- Running a suspicious file inside of controlled environment
- Monitoring various activities
- Examples: CWSandbox, Norman Sandbox, Sandboxie, SysAnalyzer,...

Motivation

- Research:



(d) Performance Overhead

[Zhiqiang Lin, Xiangyu Zhang and Dongyan Xu: Automatic Reverse Engineering of Data Structures from Binary Execution, 2010]

- Allow very close observation
- Cause massive slowdowns ($x10^3$ - $x10^4$, taint tracking)

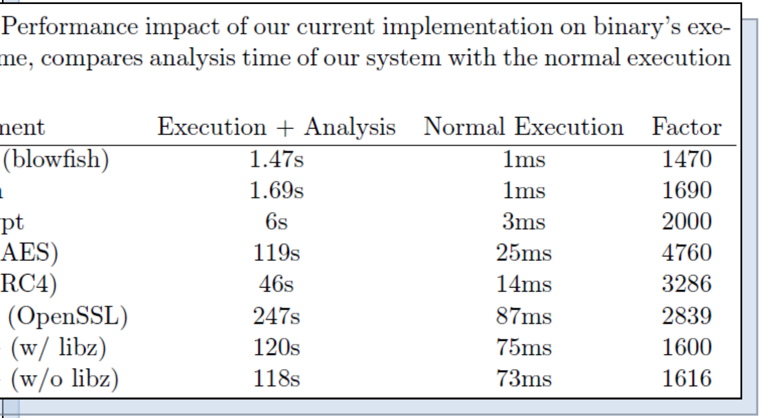


Table 4: Performance impact of our current implementation on binary's execution time, compares analysis time of our system with the normal execution time.

Experiment	Execution + Analysis	Normal Execution	Factor
Bcrypt (blowfish)	1.47s	1ms	1470
Twofish	1.69s	1ms	1690
Libgcrypt	6s	3ms	2000
cURL (AES)	119s	25ms	4760
cURL (RC4)	46s	14ms	3286
Python (OpenSSL)	247s	87ms	2839
GnuPG (w/ libz)	120s	75ms	1600
GnuPG (w/o libz)	118s	73ms	1616

[Noé Lutz: Towards revealing attacker's intent by automatically decrypting network traffic, 2008]

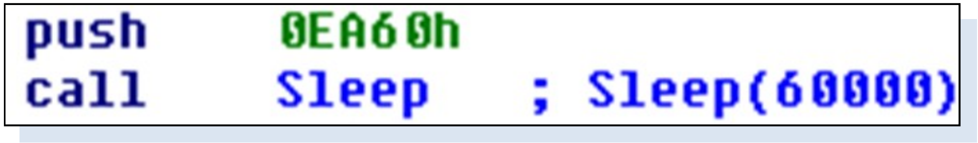
Motivation

- Existing solutions:

  e.g. Sunbelt CWSandbox

  - 500 analyzed samples/day
  - processing time/sample: **2-5 minutes**
  - Reminder: 55k samples/day
    = 1 new sample per ~1.5 sec

- Anti-Sandbox / Anti-Debugging (Storm)



```
push      0EA60h
call      Sleep     ; Sleep(60000)
```

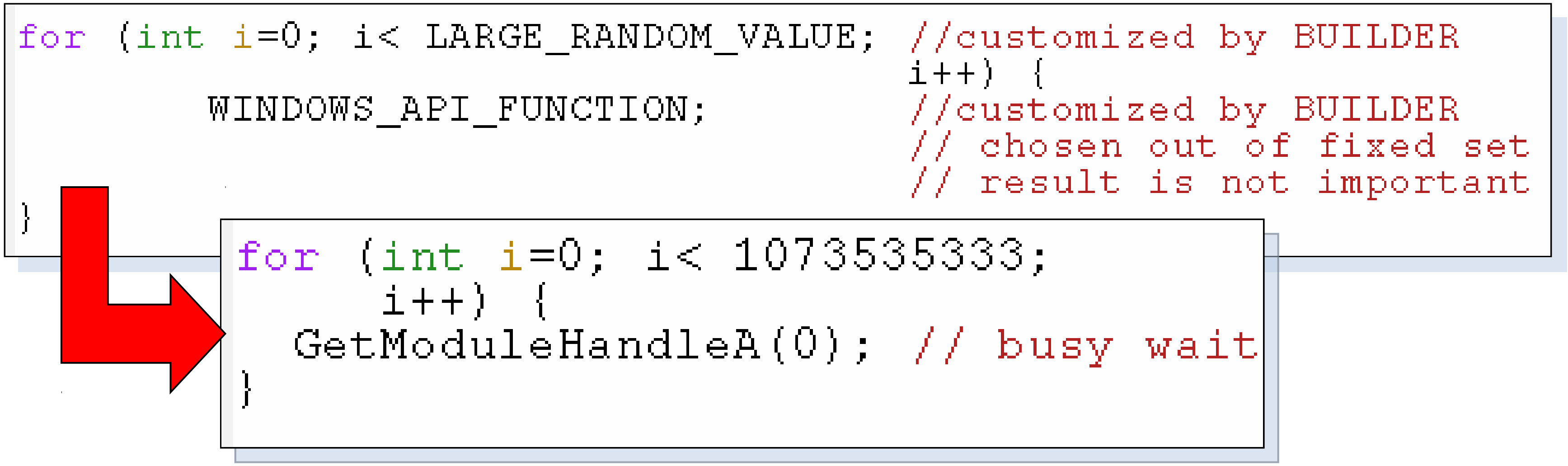
Motivation

- Existing solutions:

e.g. Sunbelt CWSandbox

- 500 analyzed samples/day
- processing time/sample: 2-5 minutes
- Reminder: 55k samples/day
  = 1 new sample per 1.5 sec

- Anti-Sandbox / Anti-Debugging (ZeuS)



```
for (int i=0; i< LARGE_RANDOM_VALUE; //customized by BUILDER
                                     i++) {
        WINDOWS_API_FUNCTION;        //customized by BUILDER
                                     // chosen out of fixed set
                                     // result is not important
}
```

```
for (int i=0; i< 107353533;
        i++) {
    GetModuleHandleA(0); // busy wait
}
```

Motivation

- Existing frameworks often have proprietary architecture
- Difficult to extend, low flexibility
- Frequent recompiling causes additional overhead

- But: fast reaction time is essential for fighting malware

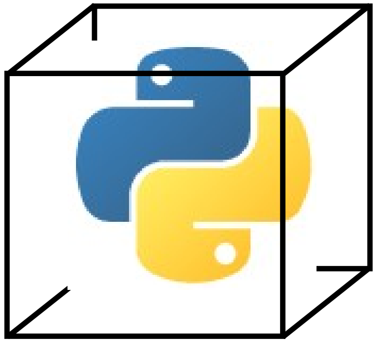
How to improve?

Use Cases:

- Need for supportive analysis tools
  - Complement dynamic / static analysis
  - Rapid prototyping for further studies
  - Enable faster malware research

- Systems forensics
  - Usage on live system
  - No special drivers or environment required

Approach

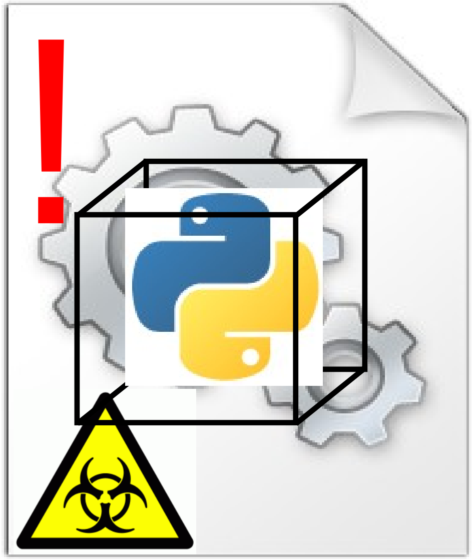


- PyBox: Toolkit for semi-automated analysis
- Major Goals: Flexibility, rapid remodelling and reconfiguration

Approach

- Basic Idea:

Inject Python interpreter into target process



Advantages

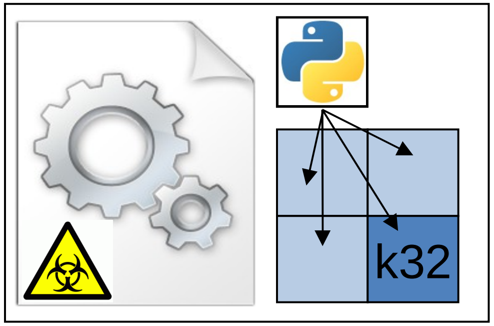
- Main functionality is extracted to scripts
  - No compiling necessary
  - Reconfiguration at runtime possible
- Modular, lightweight design
  - Easy to extend and modify
  - Dynamic management of API-hooks via scripts
  - Monitoring can be limited to relevant parts

Advantages

- Full access to registers, memory and return-values
  - Context of running process bundled via interface
  - Access provided via safety layer to avoid memory corruption
- Running on user-level
  - Ability to monitor all API calls
  - Direct reconstruction of function arguments + return-values
- Most debugger-protections do not affect PyBox
  - Exceptions, IsDebuggerPresent, Self-Debugging, …
  - And if so, adapt against it. ;)

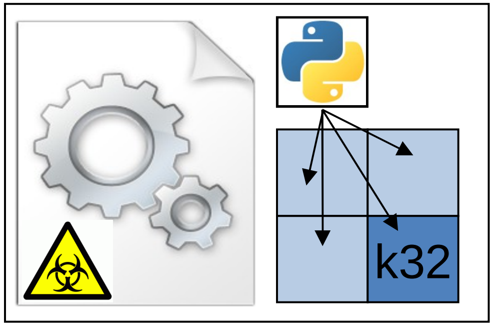
Hooking

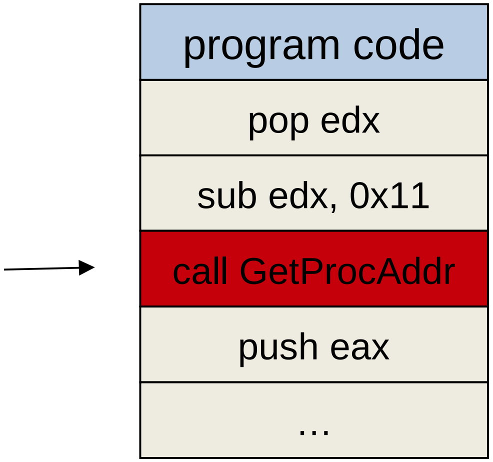
- Start suspicious file suspended
- Inject module and create hook



Hooking

- Resume main thread
- Hit a hooked function

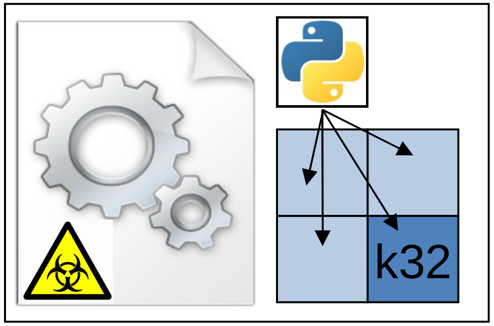


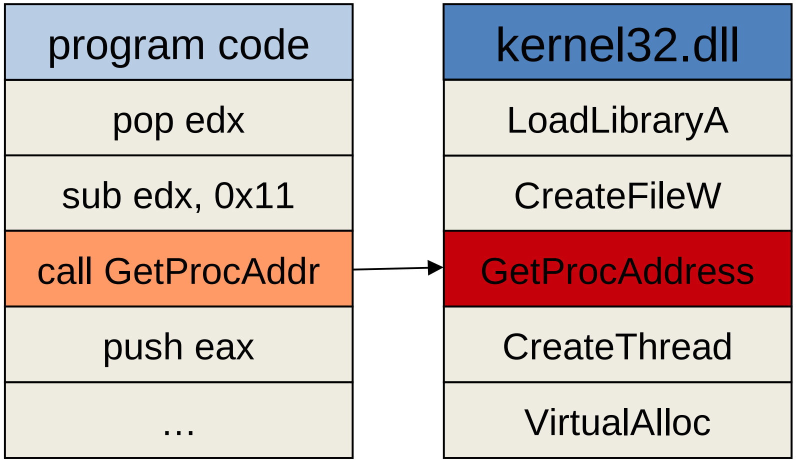


```
program code
pop edx
sub edx, 0x11
call GetProcAddr
push eax
...
```

Hooking

- call function in module
- trigger hook (jmp to a trampoline)



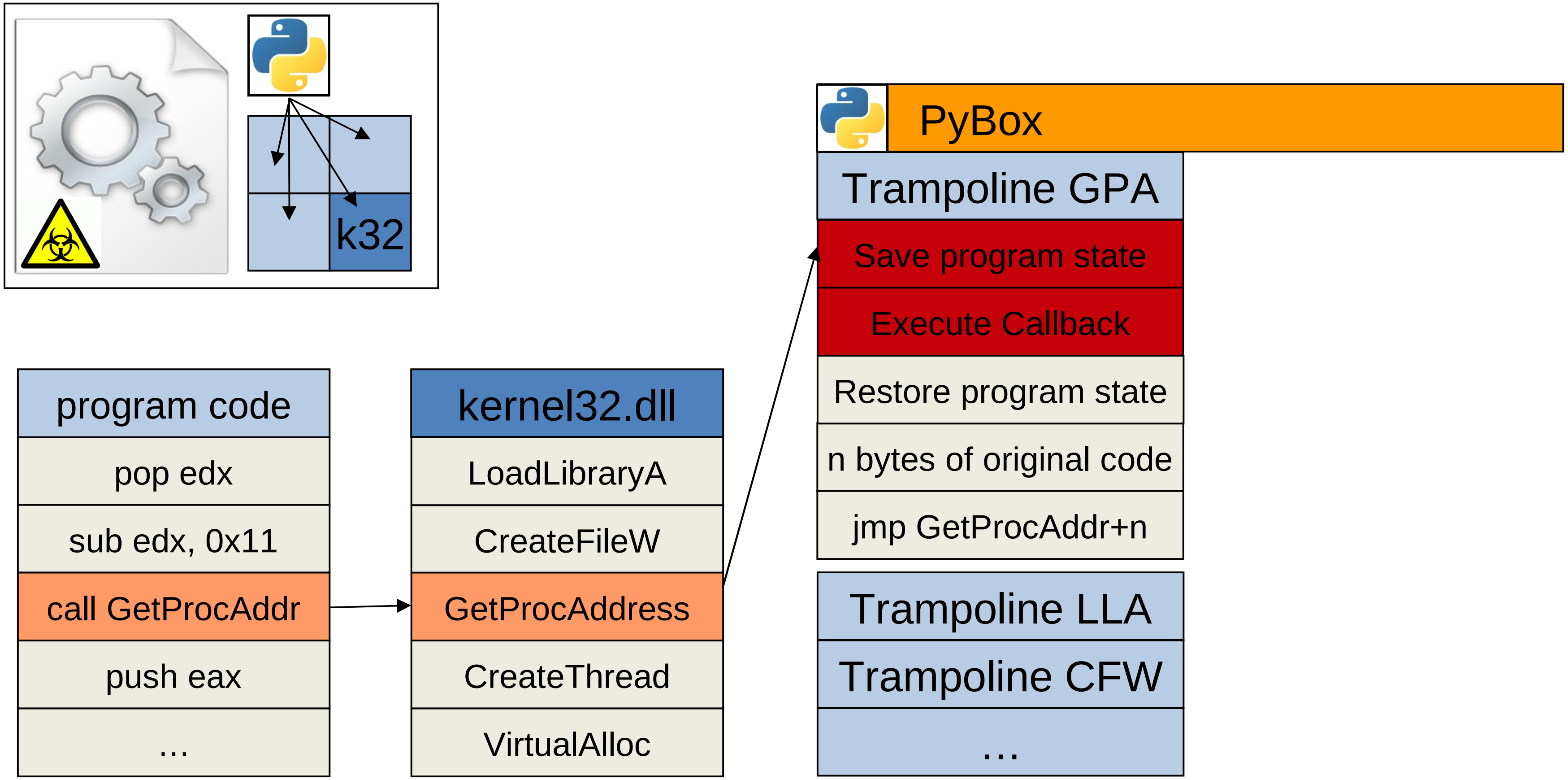


program code
- pop edx
- sub edx, 0x11
- call GetProcAddr
- push eax
- …

kernel32.dll
- LoadLibraryA
- CreateFileW
- GetProcAddress
- CreateThread
- VirtualAlloc

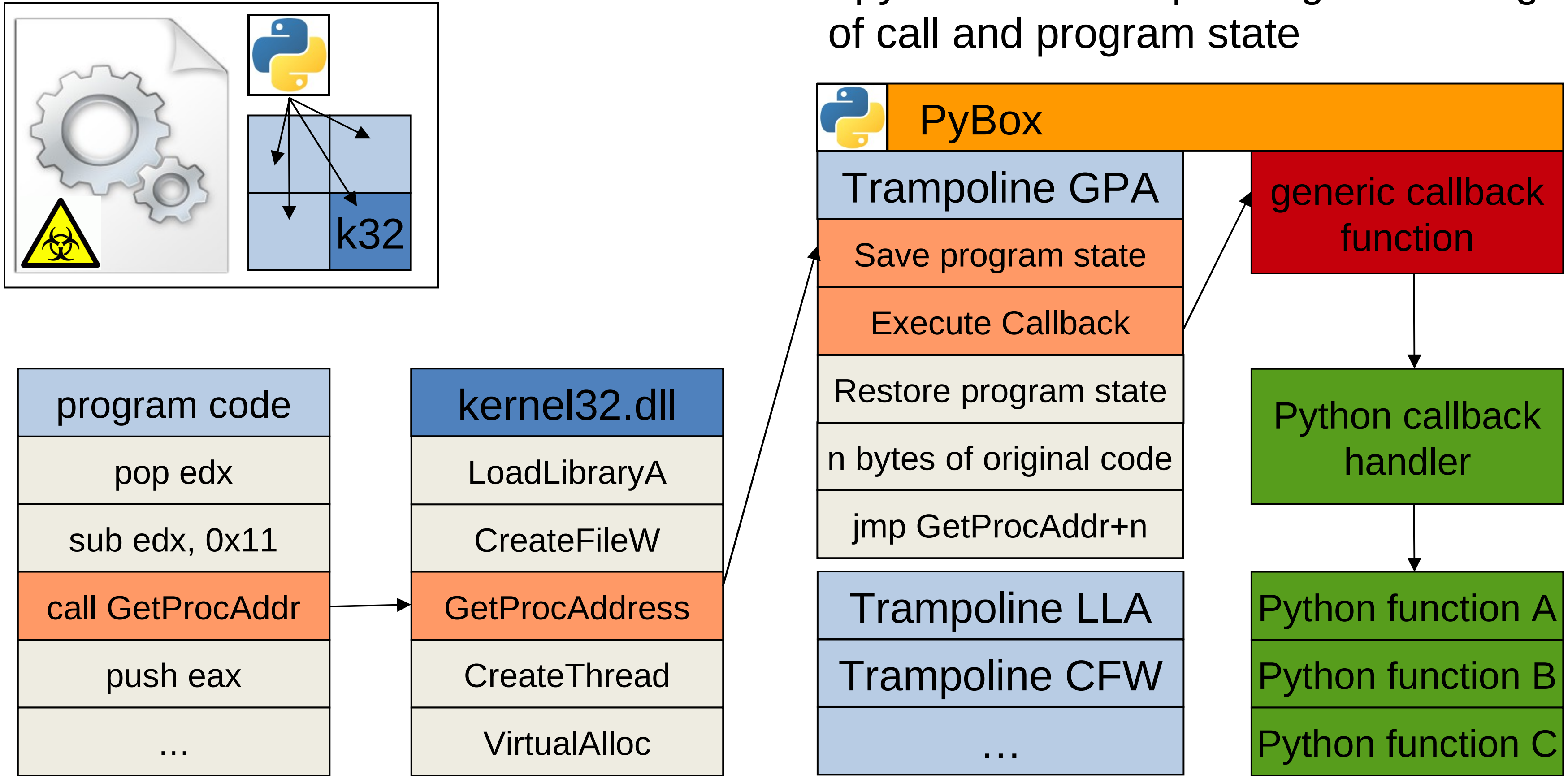
Hooking

- save process state
- execute customized callback



Hooking

- callback invokes python
- python reacts depending on the origin of call and program state



PyBox

Trampoline GPA

Save program state

Execute Callback

Restore program state

n bytes of original code

jmp GetProcAddr+n

Trampoline LLA

Trampoline CFW

…

generic callback function

Python callback handler

Python function A

Python function B

Python function C

program code

pop edx

sub edx, 0x11

call GetProcAddr

push eax

…

kernel32.dll

LoadLibraryA

CreateFileW

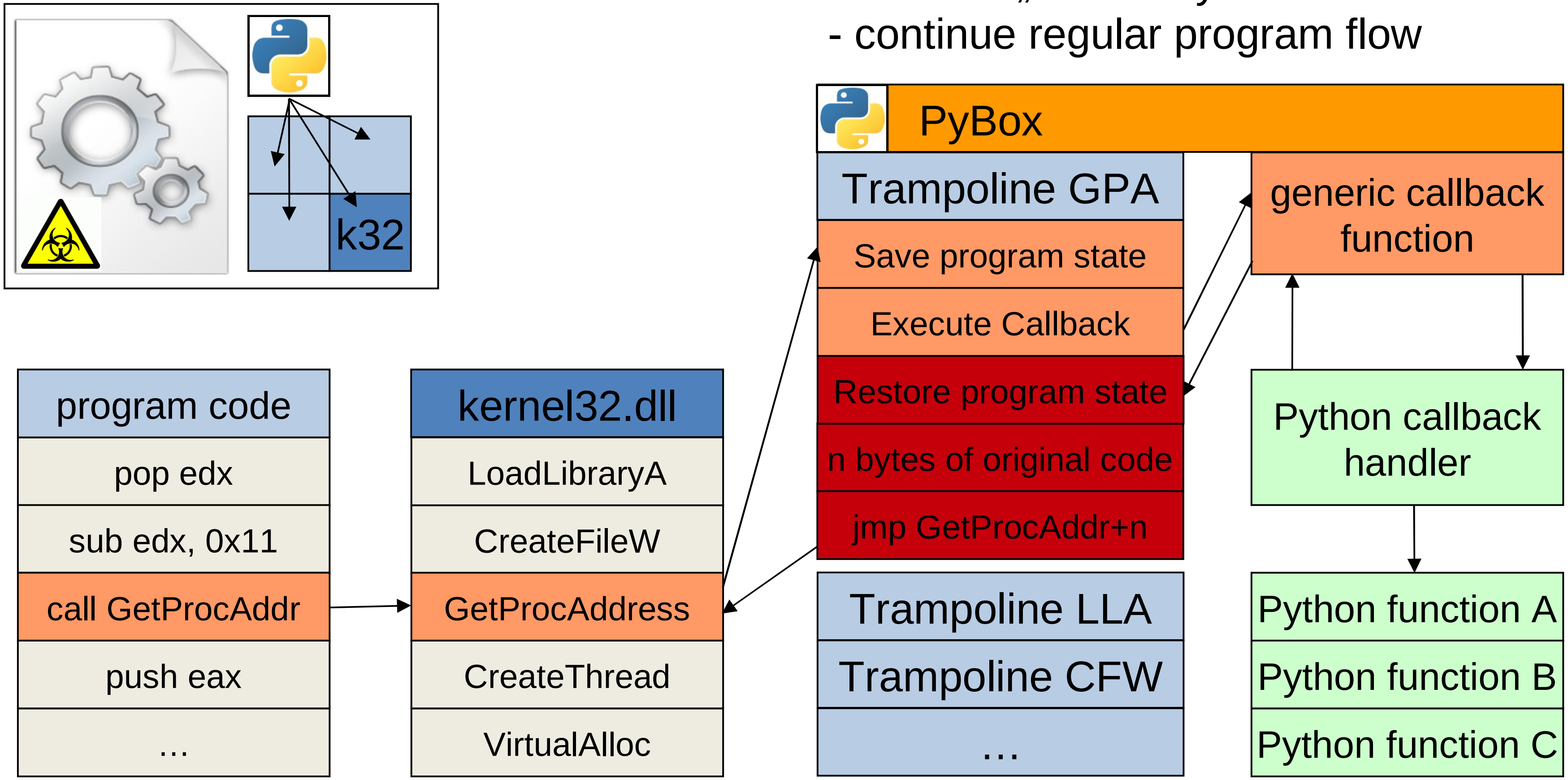
GetProcAddress

CreateThread

VirtualAlloc

Hooking

- restore original program state
- execute „stolen“ bytes
- continue regular program flow



k32

PyBox

Trampoline GPA

Save program state

Execute Callback

Restore program state

n bytes of original code

jmp GetProcAddr+n

Trampoline LLA

Trampoline CFW

…

generic callback function

Python callback handler

Python function A

Python function B

Python function C

program code

pop edx

sub edx, 0x11

call GetProcAddr

push eax

…

kernel32.dll

LoadLibraryA

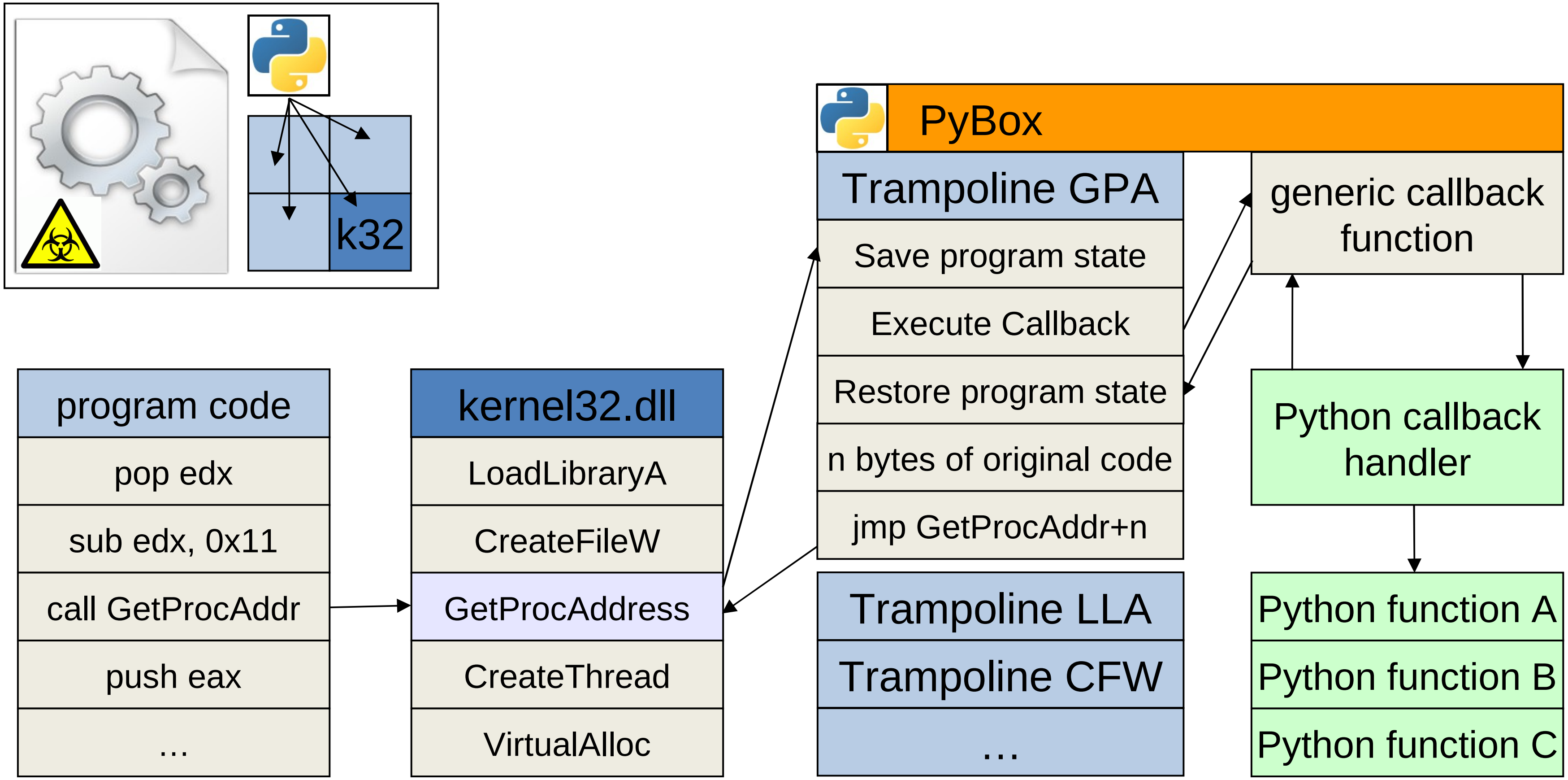
CreateFileW

GetProcAddress

CreateThread

VirtualAlloc

Hooking



program code
- pop edx
- sub edx, 0x11
- call GetProcAddr
- push eax
- …

kernel32.dll
- LoadLibraryA
- CreateFileW
- GetProcAddress
- CreateThread
- VirtualAlloc

PyBox

Trampoline GPA
- Save program state
- Execute Callback
- Restore program state
- n bytes of original code
- jmp GetProcAddr+n

Trampoline LLA

Trampoline CFW

…

generic callback function

Python callback handler

- Python function A
- Python function B
- Python function C

Live Demo

Questions?