USENIX Security’21

Obfuscation-Resilient Executable Payload Extraction From Packed Malware

Binlin Cheng*, Jiang Ming*, Erika A Leal, Haotian Zhang, Jianming Fu, Guojun Peng, Jean-Yves Marion

湖北师范大学 HUBEI NORMAL UNIVERSITY 1973

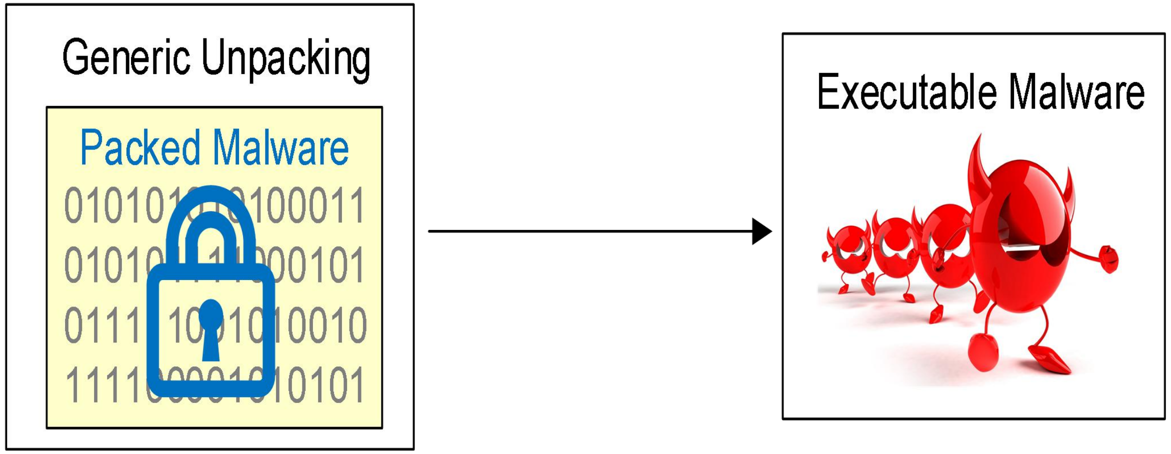
UNIVERSITY OF TEXAS ARLINGTON

武漢大學 WUHAN UNIVERSITY

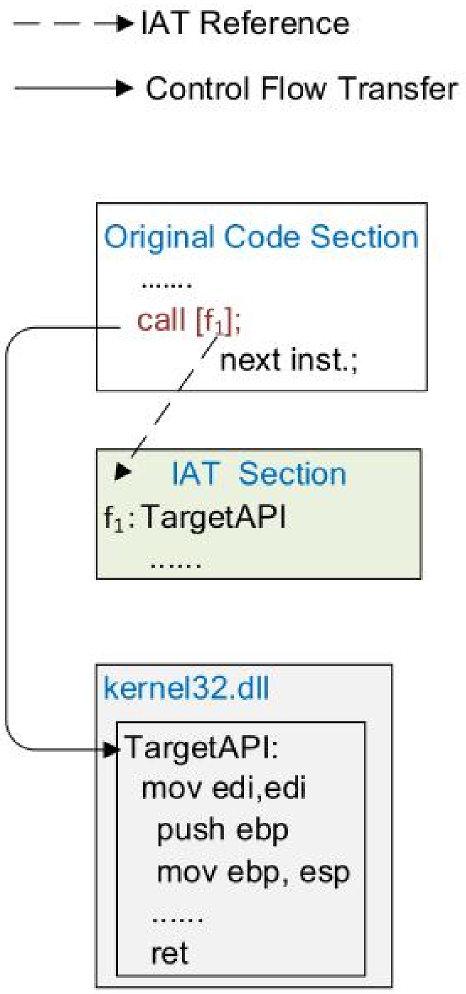
Loria Laboratoire lorrain de recherche en informatique et ses applications

This Talk is About

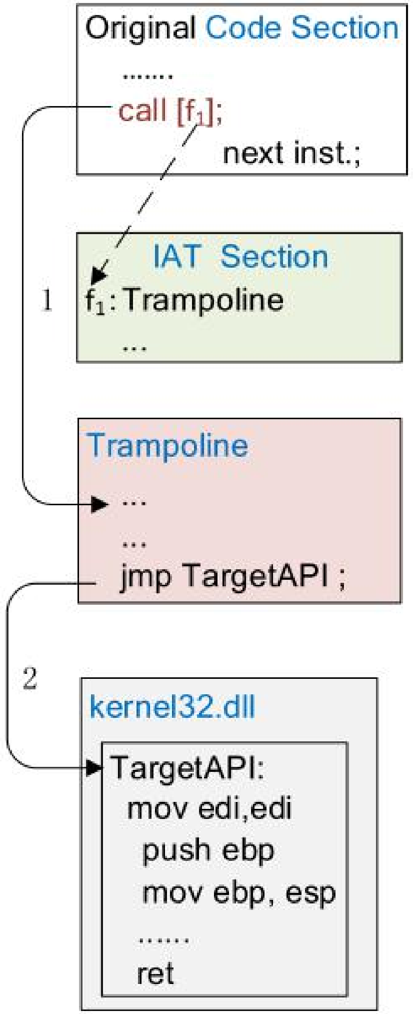
- **Restroing Executable Malware From Packed Binary Sample**



Challenge: API Obfuscation



Standard API Call



API Obfuscation

The effect of API obfuscation

- Anti-Static Analysis IDA Pro
- Anti-Dynamic Execution cuckoo

In-depth study of API obfuscation schemes

Obfuscation Type	Control Flow
Standard API Call	Original Code ⇒ TargetAPI
IAT Redirection	Original Code ⇒ Trampoline ⇒ TargetAPI
Rewrite API Callsite	Original Code ⇒ Trampoline ⇒ TargetAPI
Anti-debugging Routine	Original Code ⇒ Trampoline ⇒ Anti-debugging API ⇒ Trampoline ⇒TargetAPI
ROP Redirection	Original Code ⇒ Trampoline ⇒ End of TempAPI ⇒ Trampoline ⇒ TargetAPI
Stolen Code	Original Code ⇒ Trampoline ⇒ TargetAPI+n

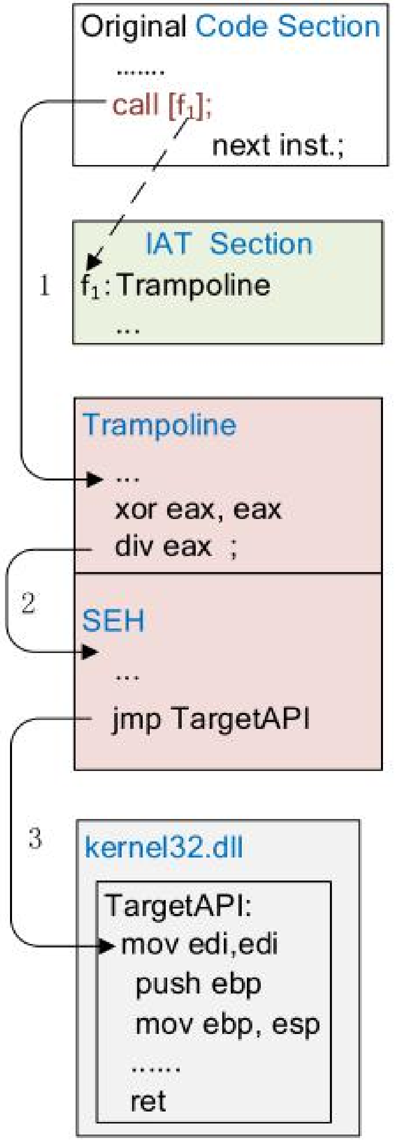
Assumptions of API de-obfuscation approaches (1)

- Assumptions 1:

  Target API' address can be statically identified in the unpacked code.

- Exception case:

  IAT Redirection via SEH:



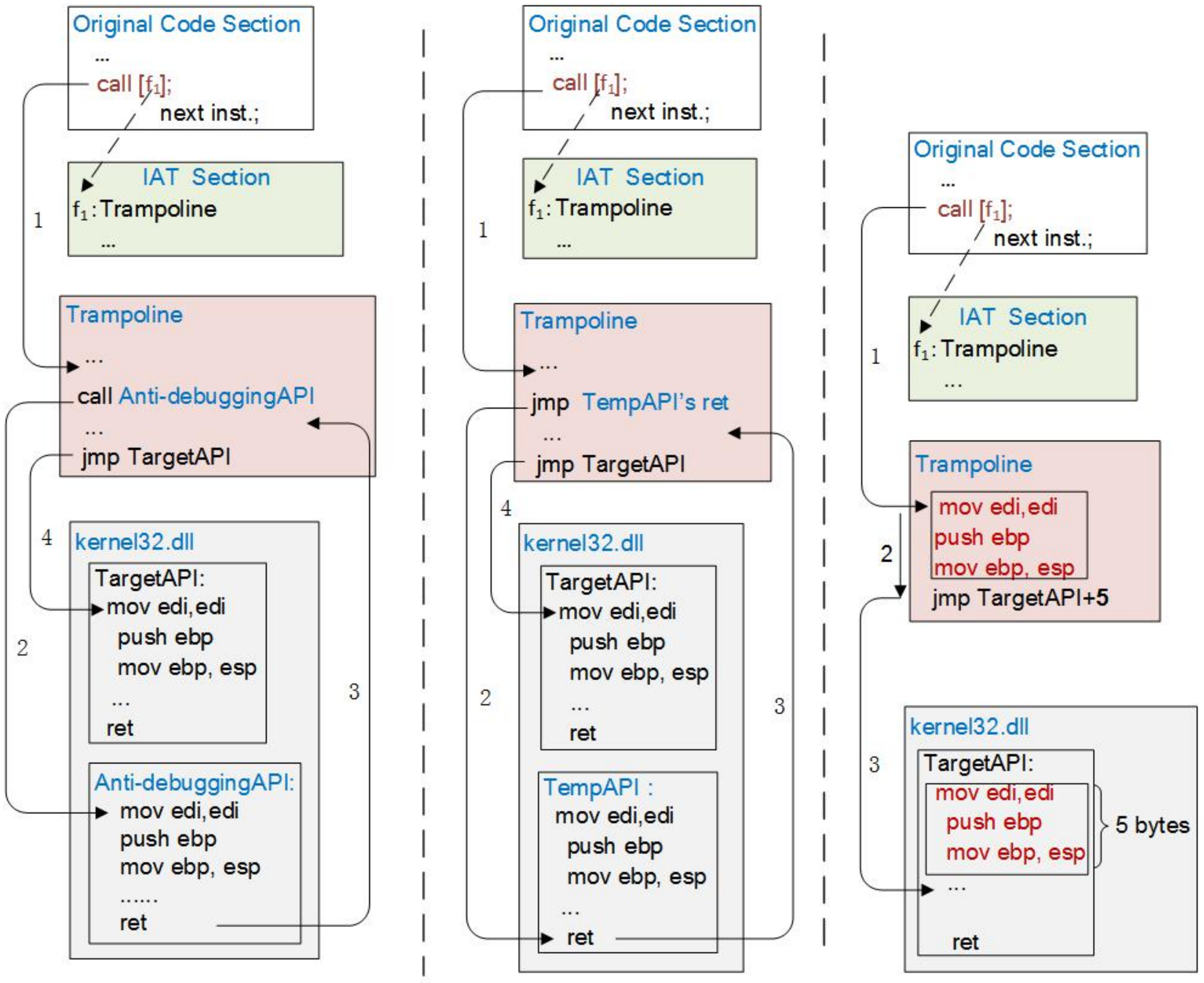
(b) IAT Redirection via SEH

Assumptions of API de-obfuscation approaches (2)

- Assumptions 2:

  When the control flow arrives at a DLL, it necessarily points to the target API's entry point.

- Exception cases：
  - Anti-debugging Routine
  - ROP Redirection
  - Stolen Code



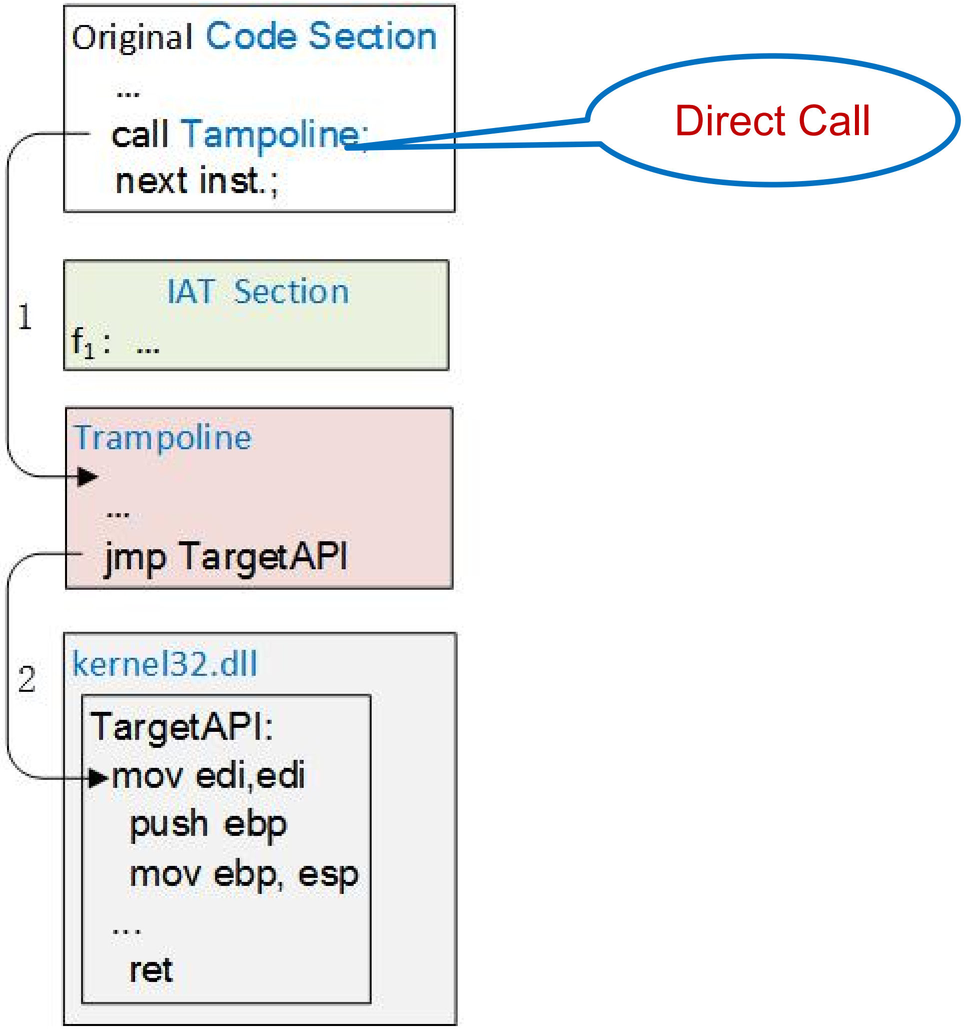
(c) Anti-debugging Routine (d) ROP Redirection (e) Stolen Code

Assumptions of API de-obfuscation approaches (3)

- Assumptions 3:

  API calls are necessarily referred to the IAT.

- Exception case：
- Rewrite API Callsite



(f) Rewrite API Callsite

Our Approach

- API-Xray: A hardware-assisted approach without any assumption.

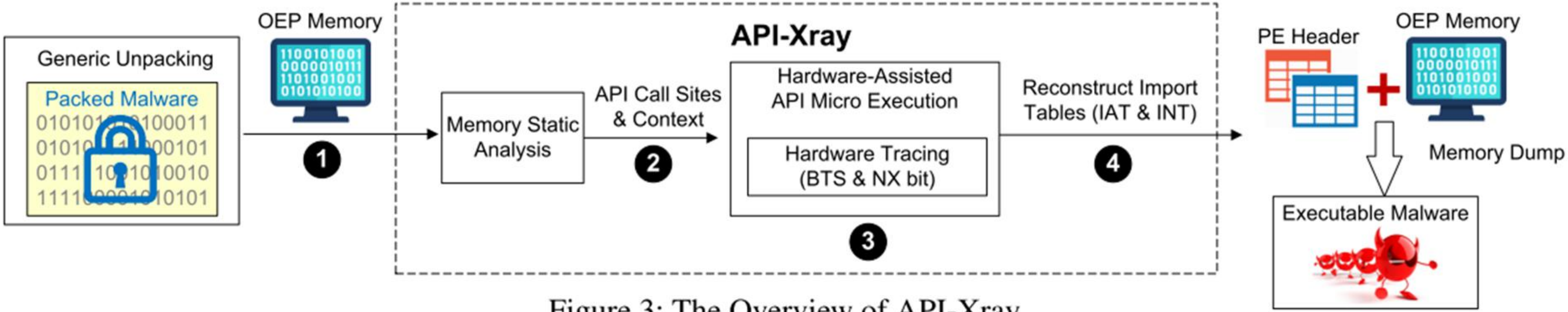


Figure 3: The Overview of API-Xray.

Hardware-Assisted API Micro Execution (1)

- **Req1**: executing the trampoline code at each API callsite;

- **Solution**: API Micro Execution.

[ICSE’14] Patrice Godefroid. Micro Execution

Hardware-Assisted API Micro Execution (2)

- **Req2:** capturing the control flow branch in trampoline, so that we can identify the target API.

- **Solution:** Intel BTS

Mechanisms	Feature
LBR	It records 16 or 32 most recent branch pairs into a register.
BTS	It records all kinds of branch pairs into a memory buffer
IPT	It does not record unconditional direct branches

The evaluation of API-obfuscation resistance

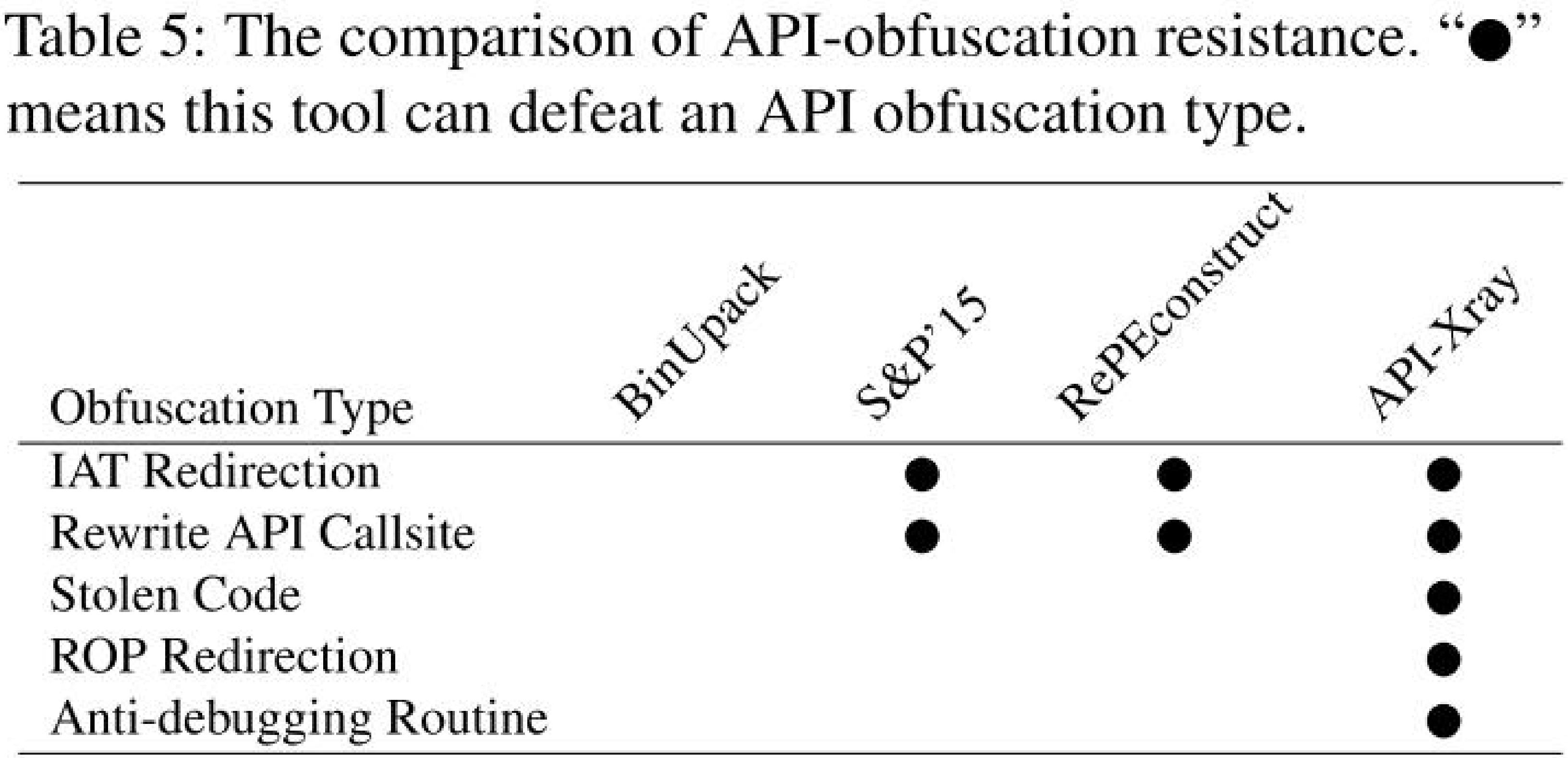


Table 5: The comparison of API-obfuscation resistance. "●" means this tool can defeat an API obfuscation type.

Obfuscation Type	BinUpack	S&P'15	RePEconstruct	API-Xray
IAT Redirection		●	●	●
Rewrite API Callsite		●	●	●
Stolen Code				●
ROP Redirection				●
Anti-debugging Routine				●

Large-Scale Evaluation

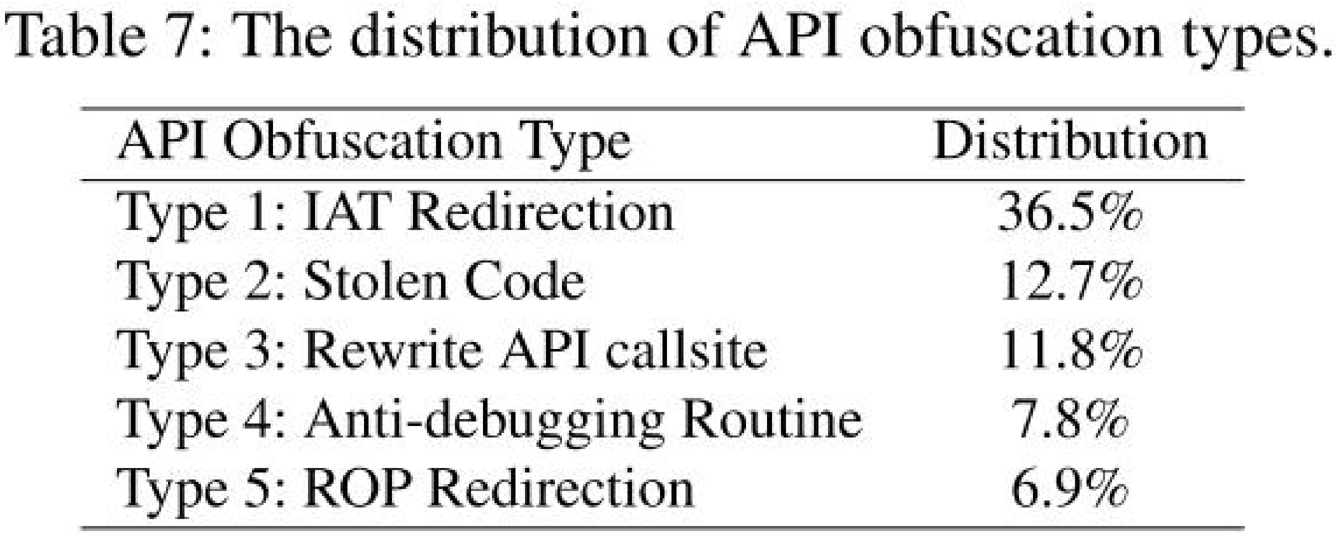


Table 7: The distribution of API obfuscation types.

API Obfuscation Type	Distribution
Type 1: IAT Redirection	36.5%
Type 2: Stolen Code	12.7%
Type 3: Rewrite API callsite	11.8%
Type 4: Anti-debugging Routine	7.8%
Type 5: ROP Redirection	6.9%

Case Study

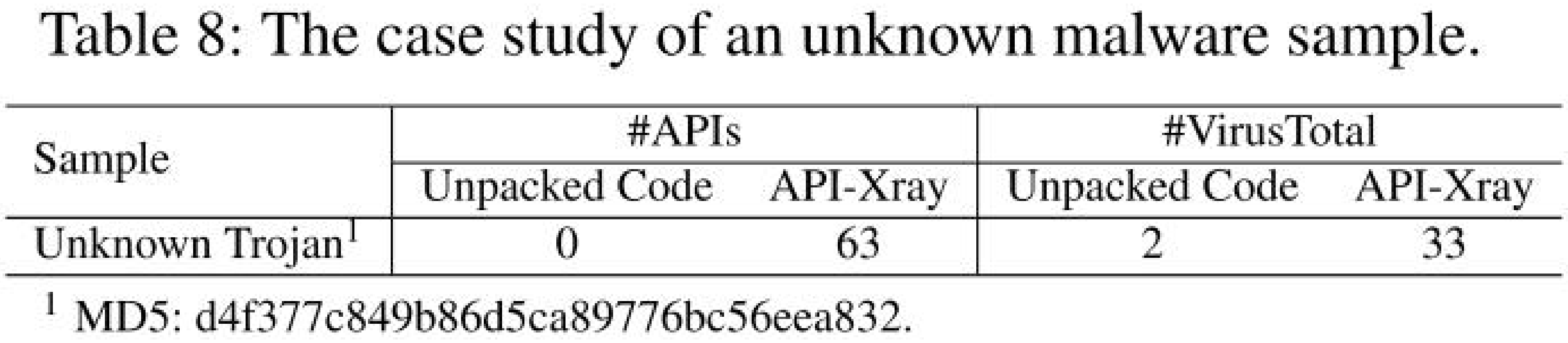


Table 8: The case study of an unknown malware sample.

Sample	#APIs		#VirusTotal	
	Unpacked Code	API-Xray	Unpacked Code	API-Xray
Unknown Trojan[1]	0	63	2	33

[1] MD5: d4f377c849b86d5ca89776bc56eea832.

Possible Attacks

- Attacks to BTS
- Attacks to NX bit.
- Statically-Linked Library
- Stolen Function.
- Argument-Sensitive Trampoline.

Please see our countermeasures in our paper!

Limitations

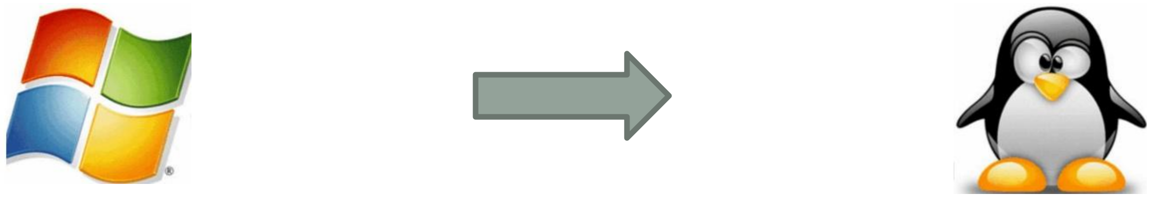
- Custom DLLs.

API-Xray cannot restore import tables from custom DLLs, which are absent in our testing environment.

- OEP Obfuscation.

Some unpacked PE files with complete import tables crashed at run time due to the OEP obfuscation.

Application to Linux Malware



- API-Xray’s technique is applied to Linux malware as well.
- That’s because API-Xray is designed to work on Intel CPU, which is independent of OS.

Q & A

Binlin Cheng (binlincheng@163.com) &
Jiang Ming (jiang.ming@uta.edu)
http://ranger.uta.edu/~ming/