From Memory Safety to Non-bypassable Security

Towards the dream of building unbreakable systems

Dr. Tao (Lenx) Wei

Chief Security Scientist, Baidu Inc.

Def Con China, 2018.5

Why Memory Safety matters?

While everybody talks about AI, Blockchain and so on

TensorFlow/Caffe/Torch Vulnerabilities

- There are many vulnerabilities in popular DL frameworks
- Consequences
  - DoS Attacks
  - Evasion attacks
  - System Compromise

DL Framework	dep. packages	CVE-ID	Potential Threats
Tensorflow	numpy	CVE-2017-12852	DOS
Tensorflow	wave.py	CVE-2017-14144	DOS
Caffe	libjasper	CVE-2017-9782	heap overflow
Caffe	openEXR	CVE-2017-12596	crash
Caffe/Torch	opencv	CVE-2017-12597	heap overflow
Caffe/Torch	opencv	CVE-2017-12598	crash
Caffe/Torch	opencv	CVE-2017-12599	crash
Caffe/Torch	opencv	CVE-2017-12600	DOS
Caffe/Torch	opencv	CVE-2017-12601	crash
Caffe/Torch	opencv	CVE-2017-12602	DOS
Caffe/Torch	opencv	CVE-2017-12603	crash
Caffe/Torch	opencv	CVE-2017-12604	crash
Caffe/Torch	opencv	CVE-2017-12605	crash
Caffe/Torch	opencv	CVE-2017-12606	crash
Caffe/Torch	opencv	CVE-2017-14136	integer overflow

[*] Security Risks in Deep Learning Implementations, Qixue Xiao et al

Python NumPy Vulnerability vs Bigdata Platforms

- Escape Python sandboxes or compromise bigdata cloud services

Quantifying the risk

It is well known that much of Python's core and many third-party modules are thin wrappers of C code. Perhaps less recognized is the fact that memory corruption bugs are reported in popular Python modules all the time without so much as a CVE, a security advisory, or even a mention of security fixes in release notes.

So yes, there are a lot of memory corruption bugs in Python modules. Surely not all of them are exploitable, but you have to start somewhere. To reason about the risk posed by memory corruption bugs, I find it helpful to frame the conversation in terms of two discrete use-cases: regular Python applications, and sandboxing untrusted code.

[*] https://hackernoon.com/python-sandbox-escape-via-a-memory-corruption-bug-19dde4d5fea5

Smart Driving - Could memory corruption kill a person?

- Unintentional acceleration by memory corruption

MADISON, Wis. — Could bad code kill a person? It could, and it apparently did.

The Bookout v Toyota Motor Corp. case, which blamed sudden acceleration in a Toyota Camry for a wrongful death, touches the issue directly.

We found a set of bugs that specifically can cause memory corruption. So they're lurking there. And if they happen, then as a result of that, then the some critical variable could be -- could have a new value, for example, the throttle commend could become instead of opening 20 percent opening 50 percent letting in a lot more air and giving the engine a lot more power.

[*] https://www.eetimes.com/document.asp?doc_id=1319903

Pwn2Own Every Year

GeekPwn Every Year

Towards a dream of building unbreakable systems

- Defense-in-depth is an effective strategy for enterprise security
  - Vulnerabilities and exploits can be contained in multiple defenses
- But AI-based systems demand far better security
  - No enough defense-in-depth for smart home devices
  - Unwanted crash is an effective mitigation for browsers against exploits, but unwanted crashes might be fatal in a self-driving system

Memory Safety

And unsafety

Memory safety

- Memory safety is the state of being protected from various software bugs and security vulnerabilities when dealing with memory access, such as buffer overflows and dangling pointers. [1]
- Spatial error [2]
  - Dereferencing an out-of-bounds pointer
- Temporal error [2]
  - Dereferencing a dangling pointer

[1] https://en.wikipedia.org/wiki/Memory_safety

[2] SoK: Eternal War in Memory, Laszlo Szekeres, Mathias Payer, Tao Wei, Dawn Song, S&P 2013

Memory safety in Programming Languages

- Most of modern high-level languages are designed with memory-safe features.
  - Rust
  - Go
  - Swift
  - Javascript
  - Python
  - Java
  - Ethereum Solidity
  - ……

Memory Unsafety in “Memory-safe” Programming Languages

- Java’s JVM
- Python’s C libs
- Swift’s Object-C runtime
- Javascript’ GC engine
- Go’s asm code
- Rust’s unsafe
- ……

Practical Memory Safety

- Pure memory safety is impractical for real world applications today
- Practical Memory Safety is layered & hybrid
- **3 principals** for a hybrid memory-safe architecture

  [Proposed in Baidu X-Lab Rust SGX SDK]
  - Unsafe components must not taint safe components, especially for public APIs and data structures
  - Unsafe components should be as small as possible and decoupled from safe components
  - Unsafe components should be explicitly marked during deployment and ready to upgrade

Practical Memory-safe Projects by Baidu X-Lab

- Rust SGX SDK: Write Intel SGX applications in Rust
- MesaLock Linux: A Memory-Safe Linux Distribution

MesaLock

- MesaLink: A Memory-safe OpenSSL-compatible TLS Library

MesaLink

- And more to come soon

Re: Why Memory Safety is important

- Memory unsafe programs contain hidden control flow/data flow by breaking memory boundaries
  - The analysis cost becomes too high to be practical for real-world applications
- Memory safety makes control flows and data flows explicit
  - Security audit is much easier
    - Web/Android Java audit vs. Windows/Linux binary audit
  - Classical formal verification has not fully made use of this feature yet

Non-bypassable Security Paradigm (NbSP)

Towards formal verification of security properties of control flows

Is Memory safety Hackproof?

- Just one step further, but not bullet-proof
- Control-flow hijacking is still possible
  - Android JavaScript Bridge
  - Java Reflection abuse
  - Struts2 OGNL vulnerabilities
  - ……
- Data-flow vulnerabilities are still possible
  - Sql Injections
  - Solidity Integer overflow
  - ……

Case study: Control-flow hijacking

- SIDEWINDER TARGETED ATTACK AGAINST ANDROID IN THE GOLDEN AGE OF AD LIBRARIES
  - Yulong Zhang, Tao Wei, Blackhat 2014
  - Use popular ad libs to intercept location information, opening the door to targeting specific areas (say, a CEO's office), and then take photos or record videos remotely
  - Android JavaScript-biding-over-http + Java reflection abuse
  - No memory-unsafe vulnerability exploited

```
jsObj.getClass().forName(”java.lang.Runtime”)
.getMethod(”getRuntime”,null).invoke(null,null).exec(cmd)
```

From Memory Safety To Formal Verification

- Machine-checkable formal verification is the only theoretically unbreakable hackproof methodology today, the holy grail
  - But the cost is too high to fully verify most of real world applications
  - Layered formal verification is a promising direction
- Memory Safety
  - Make control flows and data flows explicit, but
  - Control-flow hijacking is still possible
  - Data-flow vulnerabilities are still possible
- Non-bypassable Security Paradigm (NbSP)
  - Based on memory safety
  - Layered formal verification of security properties of control flow

Non-bypassable Security

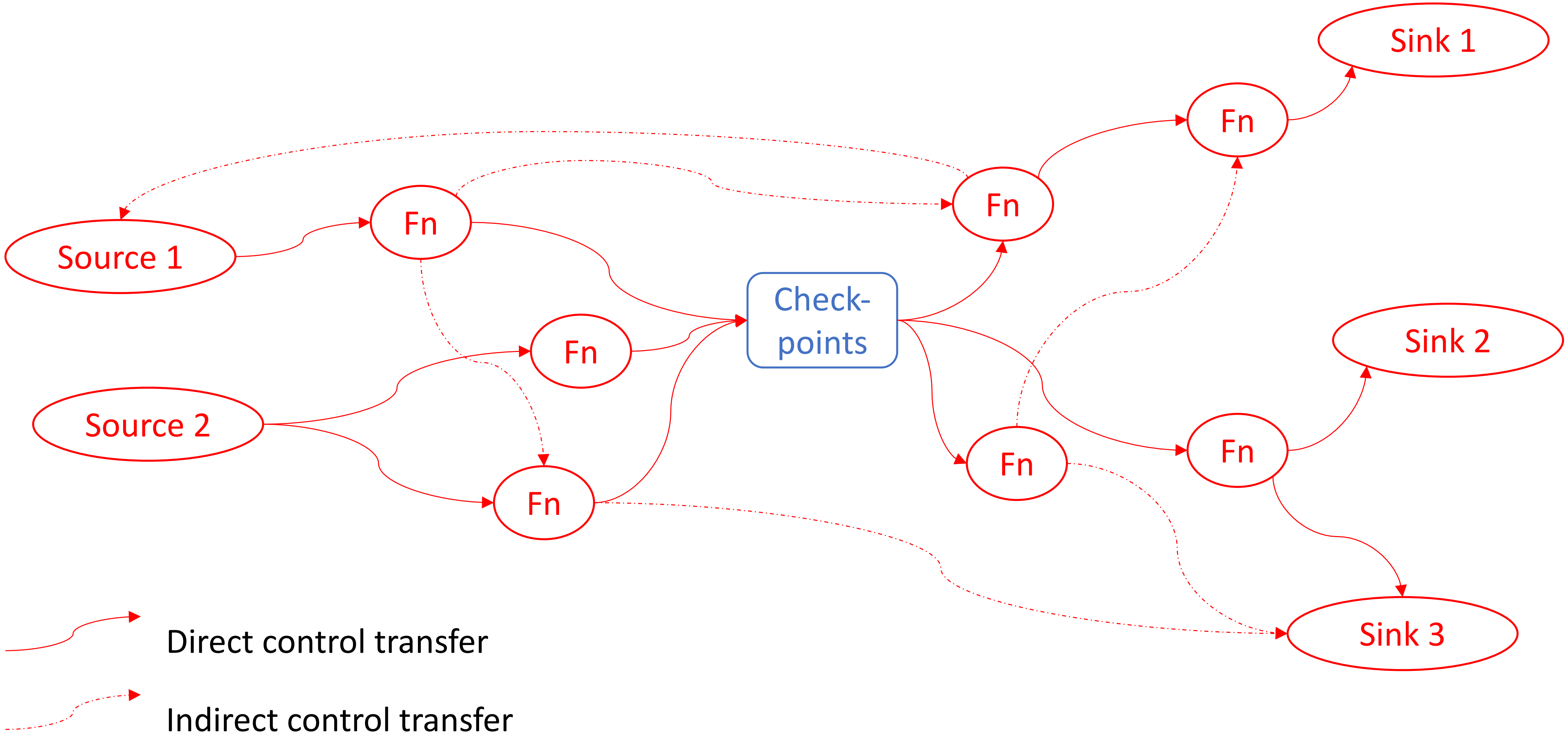
- Introduced by MILS (Multiple Independent Levels of Security/Safety)
- It requires that one component cannot use another communication path, including lower level mechanisms to bypass the security monitor
- Critical security checkpoints should be guaranteed to be non-bypassable
  - Authentication
  - Authorization
  - Auditing
  - ……

Non-bypassable Security Paradigm

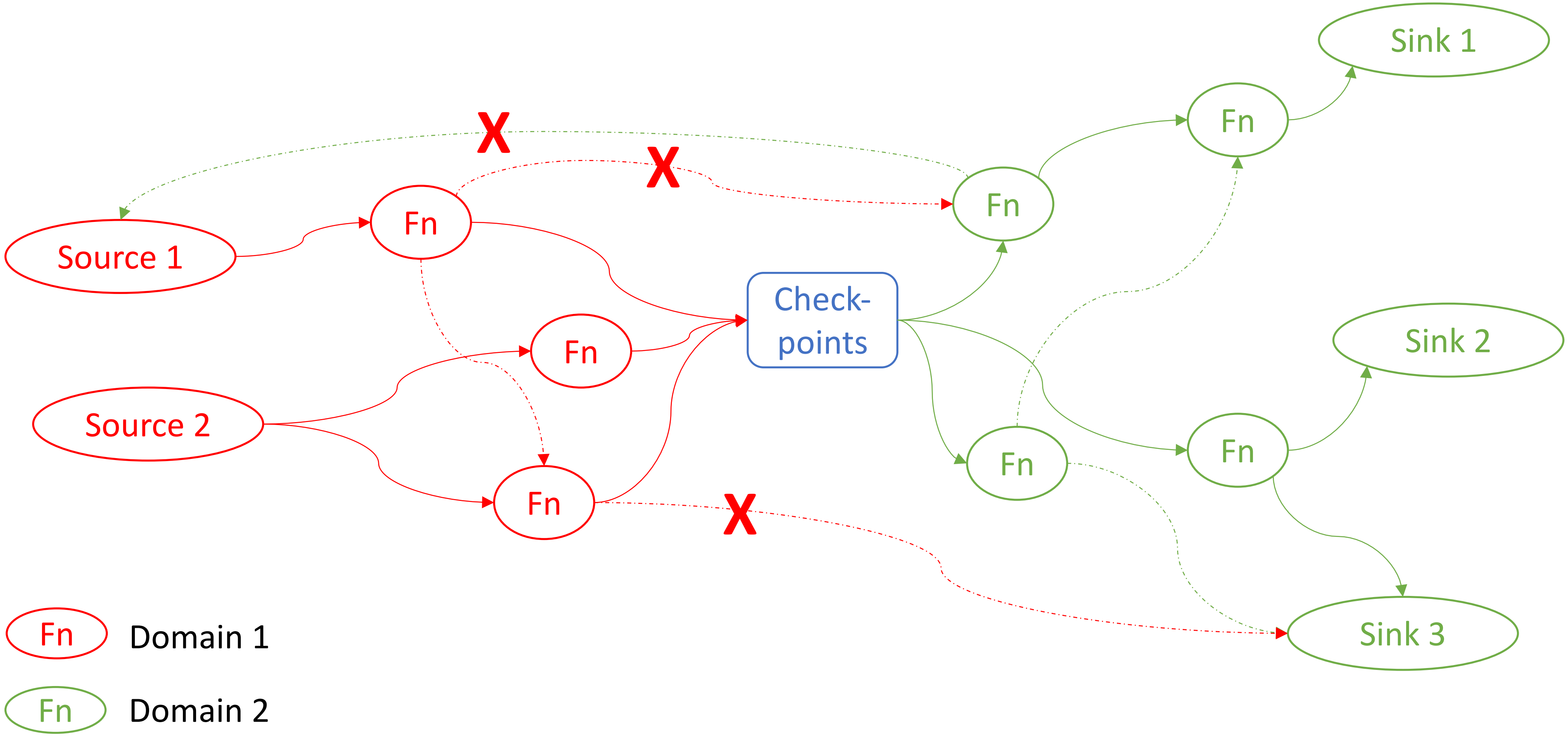
Control-flow Formal Verification for Memory-safe Applications

- We just need to make sure that all the paths between sources (e.g. input) and sinks (e.g. database operations) MUST contain critical security checkpoints
  - Formal machine-checkable verifications
- Advantages
  - Straight-forward for direct control flows
  - Control flow graph is explicit, and we don't need to dig hidden control flows generated by memory unsafety
- Challenges
  - Alias analysis: function pointers, reflections and so on
  - Can not guarantee both soundness (no FN) and completeness (no FP) at the same time

Indirect-bypassable challenges



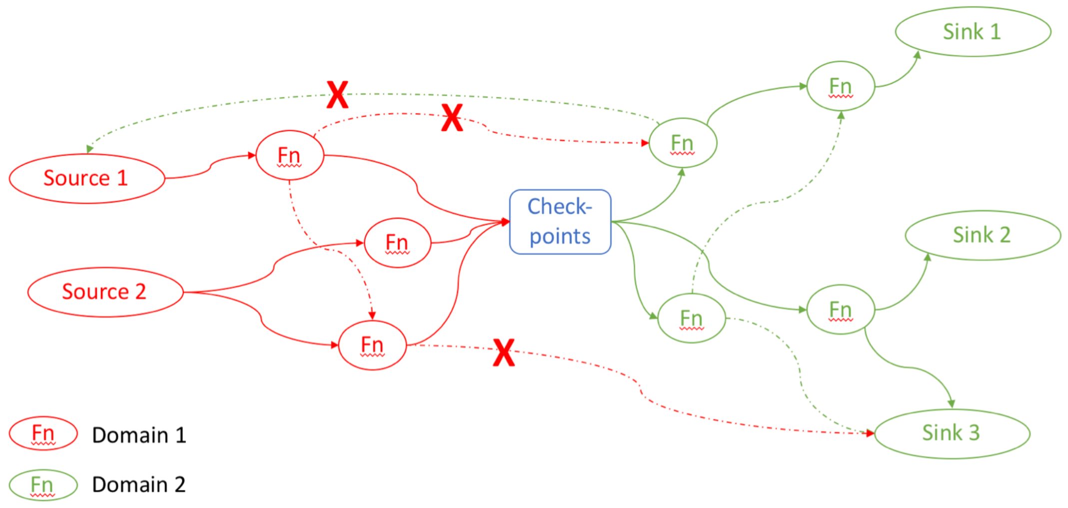
Use different types to block potential bypass



Non-bypassable Security Paradigm

A Sound Solution (no FN)

- Machine-checkable formal verification
  - Explore all the potential control flows in a given memory-safe program, including direct and indirect ones
  - Reject if any path between a source and a sink bypasses given checkpoints
- Software design and implementation
  - Remove unnecessary direct bypasses, or add missed checkpoints
  - Use different function types to block unwanted indirect bypasses



Non-bypassable Security Paradigm

Towards full formal verification of security checkpoints

- All the paths towards checkpoints are known
- Data-flow formal verification for all these paths is the next step, against
  - Sql Injections
  - Integer overflow
  - ……
- A practical layered solution of formal verification
  - Depend on layered memory safety assumptions

Conclusions

Towards the dream of building unbreakable systems

From Memory Safety To Non-bypassable Security Paradigm

- Memory Safety and software security is even more important along with Cloud, AI, Blockchain, IoT, Smart driving ...
- Memory Safety makes control flows and data flows explicit
  - The pre-condition of *sound* verification (no false negatives)
- Non-bypassable Security Paradigm (NbSP) guarantees that critical checkpoints are non-bypassable, i.e. no missed/hidden paths
  - A practical trade-off between design/implenmentation and analysis completeness
  - Machine-checkable formal verification of security properties of control flows
- NbSP reduces attack surfaces significantly to critical checkpoints
  - Authentication, authorization, auditing and so on
  - NbSP makes it practical for further formal verification (data flows)

Welcome to join our open-source memory-safe projects

- Rust SGX SDK: Write Intel SGX applications in Rust
- MesaLock Linux: A Memory-Safe Linux Distribution
- MesaLink: A Memory-safe OpenSSL-compatible TLS Library
- And more to come soon

Thanks!

Dr. Tao (Lenx) Wei

Chief Security Scientist, Baidu Inc.

Def Con China, 2018.5