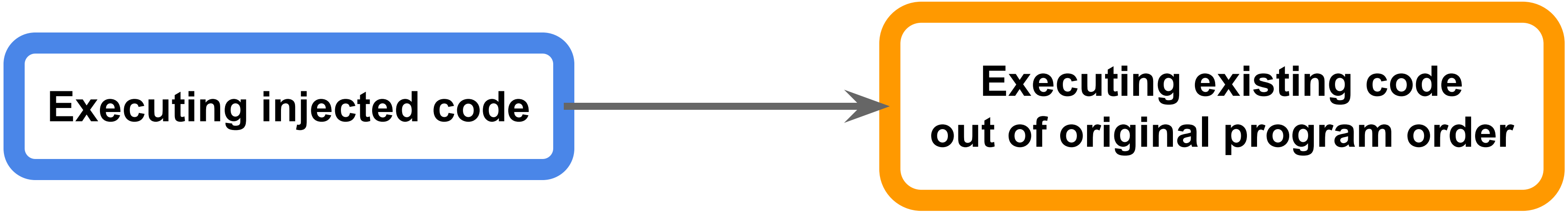
Abusing Performance Optimization Weaknesses to Bypass ASLR

Byoungyoung Lee
Yeongjin Jang
Tielei Wang
Chengyu Song
Long Lu
Taesoo Kim
Wenke Lee

Georgia Tech Information Security Center

(Rough) System Attack Trends



Executing injected code

- Ret/Jmp/Call to Stack
- Ret/Jmp/Call to Heap

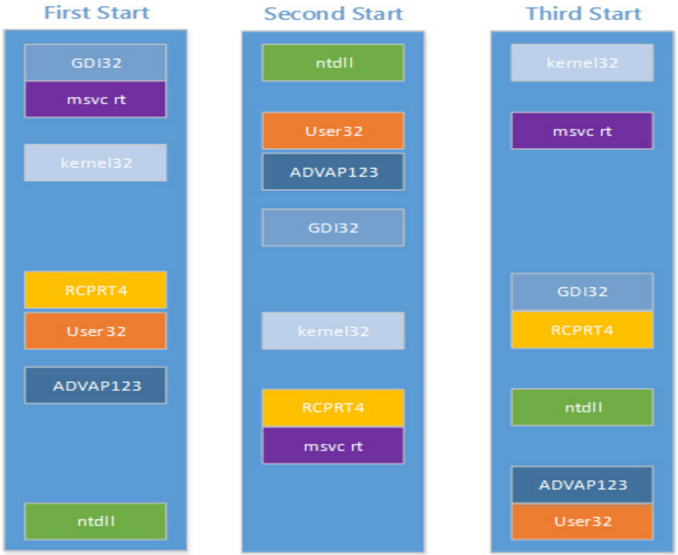
Executing existing code out of original program order

- Ret/Jmp/Call to libc
- Ret/Jmp/Call to “gadgets”

In general, to launch such attacks, one needs to know the addresses of stack, heap objects, code gadgets, etc.

Address Space Layout Randomization (ASLR)

- Intuition: introducing diversity into the memory layout of computer systems will defeat many easily replicated attacks [1]



http://technet.microsoft.com/en-us/library/dn283963.aspx

A Brief History of ASLR

Stack Randomization	PaX Project	Linux Kernel 2.6.12	MS Vista Mac OS X Leopard 10.5	iOS 4.3 Android 4.0
1998/1999	2001	2005	2007	2011

Bypassing ASLR

- Abusing non-randomized data structures
  - Executables compiled without the PIE flag
  - VirtualAlloc and MapViewOfFile are not randomized [2]
  - SharedUserData is located at fixed address[3]

Bypassing ASLR

- Abusing low entropy
  - Heap Spraying and JIT Spraying
  - Effective brute force attacks against fork-only servers [4]

Bypassing ASLR

- Exploiting vulnerabilities to leak addresses
  - Type Confusion
  - Heap Overflow
  - Use-after-free
  - Integer Overflow
  - Format String
  - Uninitialized Memory Read

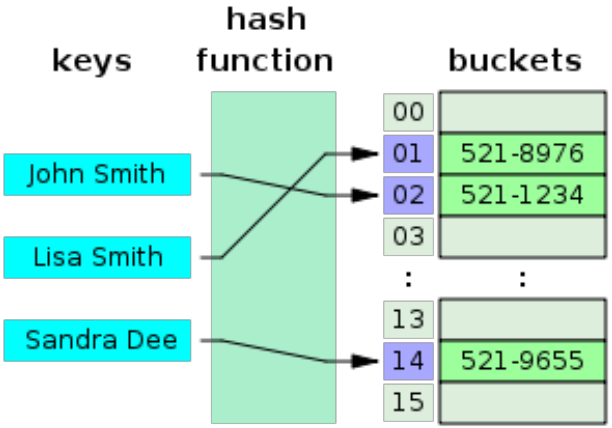
Today, we will talk about ...

- Performance oriented designs that are at odds with ASLR


julia
Java

Hash Table

- Hash Table
  - map **keys** to **values**
  - **keys** are hashed to find proper **buckets**



bucket# = hash(key) % arraySize

Hash Table

- Collision Resolution
    - # of buckets are limited!
    - Open addressing: find the next available bucket
        - Linear probing
        - Quadratic probing
        - Double hashing

Hash Table and ASLR?

- Built-in hash tables
  - JavaScript, Java, Python, Ruby, …
  - Sometimes they use memory addresses as a key for objects

Hash Table and ASLR?

- Memory addresses as a key
  - (Always) unique identifier for an object
  - Fast / Easy to implement

- Alternatives
  - Random numbers ⇒ collision free?
  - Static counters ⇒ thread safe?

Hash Table and ASLR?

Q. Can you read the key?

A. If the language allows, yes

```
x = object()
id(x)
hash(x)
```

Q. Is this a security breach?

A. It depends

Address Information in Script Languages

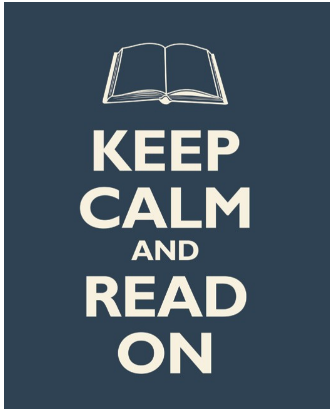
- Usually running scripts from the shell means you have everything.

- What if it is running in restricted environments?
  - Sandboxed environments
  - Many script languages have sandbox-like extensions for

Hash Table and ASLR?

Q. Can you still read the key even if it is not allowed?

A. Partially, via timing attacks



Attacking ASLR with Hash Tables

	Can you read the key?	Can you infer the key?	Is the key a memory address?
Python	yes	-	yes
Ruby	yes	-	yes
Julia	yes	-	yes
PHP	yes	-	no
Java (JVM)	yes	-	no
Java (DVM)	yes	-	yes
JavaScript (WebKit)	no	yes	yes
JavaScript (V8)	no	yes	no

Examples - Directly Reading a Key

```
x = object()
id(x)
hash(x)
```

```
x = object()
x.object_id
```

```
Object x = new Object();
x.hashCode();
```

```
type x
end
object_id(x)
```

Hash Table in WebKit JavaScript

- Name object
  - Adding (unique) private properties to any object
  - New (experimental) features for ES6 Harmony
  - How is it unique?
    - using memory addresses

```
// Source/WTF/wtf/text/StringImpl.h

enum CreateEmptyUniqueTag { CreateEmptyUnique };
    StringImpl(CreateEmptyUniqueTag)
        : m_refCount(s_refCountIncrement)
        , m_length(0)
        , m_data16(reinterpret_cast<const UChar*>(1))
    {
        ASSERT(m_data16);

        unsigned hash = static_cast<uint32_t>(reinterpret_cast<uintptr_t>(this));
        hash <<= s_flagCount;
        if (!hash)
            hash = 1 << s_flagCount;
        m_hashAndFlags = hash | BufferInternal;

        STRING_STATS_ADD_16BIT_STRING(m_length);
    }
```

How to Infer a Key in WebKit Javascript

- Requirements
  - Collision resolution should follow a certain order
  - A hash algorithm must be deterministic
  - Hash tables must be (partially) controllable

How to Infer a Key in WebKit Javascript

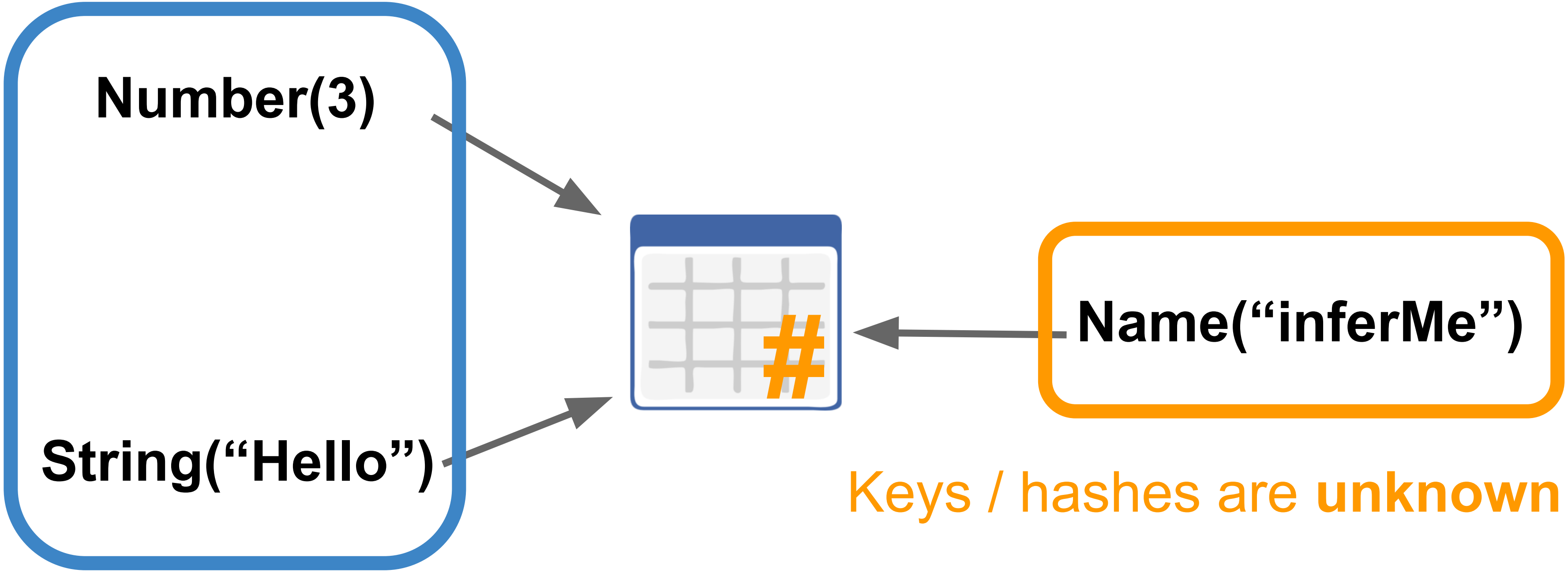
- Requirements **(in WebKit JavaScript)**
    - Collision resolution should follow a certain order

      ⇒ **Double hashing**
    - A hash algorithm must be deterministic

      ⇒ **Yes**
    - Hash tables must be (partially) controllable

      ⇒ **Number / String**

How to Infer the Key in WebKit Javascript



Number(3)

String("Hello")

Name("inferMe")

Keys / hashes are **unknown**

Keys / hashes are **known**

Abusing Collision Resolution

- Linear Probing
    - If there’s a collision, simply try the next slot
    - Given key k & i^{th} trial

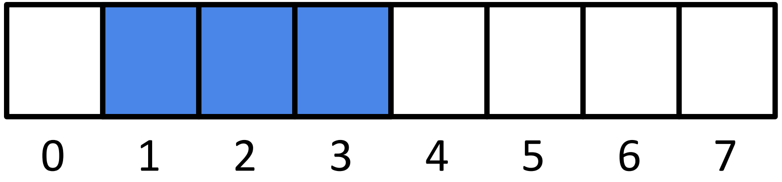
$$\text{bucket\#} \Leftarrow (\text{hash}(k)+i) \mathbin{\%} \text{tableSize}$$

Abusing Collision Resolution

bucket# ⇐ Hash(Number(1)) % 8 = 1

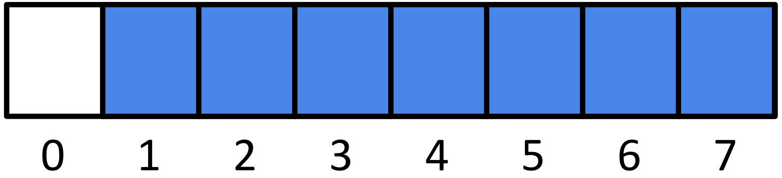
bucket# ⇐ Hash(Number(9)) % 8 = 1

bucket# ⇐ Hash(Number(17)) % 8 = 1



Abusing Collision Resolution

- Search timing differences
    - Assume the table is filled up
    - Except bucket # 0

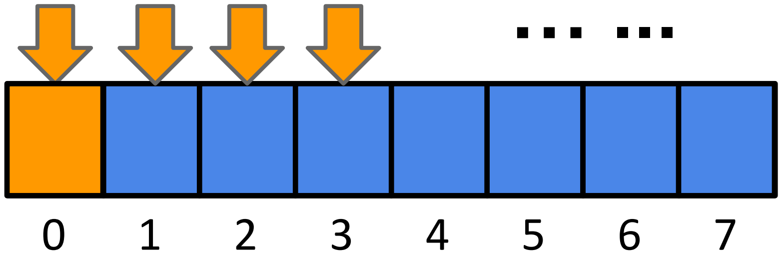


Abusing Collision Resolution

- Time differences b/w worst and best cases

bucket# ⇐ Hash(Number(1)) % 8 = 1

⇒ **Slow**

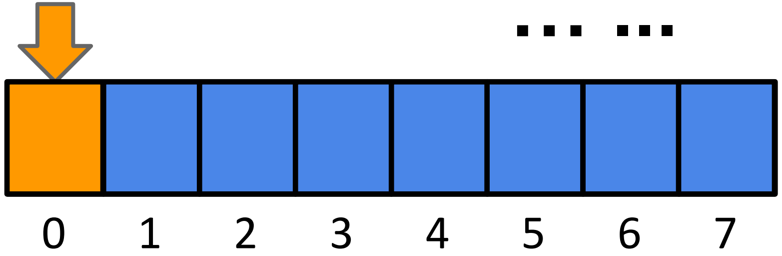


Abusing Collision Resolution

- Time differences b/w worst and best cases

bucket# ⇐ Hash(Number(0)) % 8 = 0

⇒ **Fast**



Abusing Collision Resolution

- If you catch this timing difference,
  ⇒ learn something about hash (key)
  ⇒ one bit information at a time

- Is this doable in JavaScript?
  - JS timer is mili-seconds ⇒ repeat thousand times
  - Table size is too big ⇒ repeat a lot again

  ⇒ **Calibration !**

Abusing Collision Resolution

- **Weak key**
  - A key that the second hash function returns small integer numbers
    - To avoid fuzziness of double-hashing
  - Can be found with high probabilities

⇒ Repeat the Name object creation until we find the weak key

Abusing Collision Resolution

- Prototype implementations
  - Ported WebKit's hash functions to JavaScript
  - Pre-built an inversion table
    - h^{-1}(String/Number) ⇒ bucket #
  - Currently leaking 12-bits
    - Possible up to 23-bits
    - Need better mathematical properties.

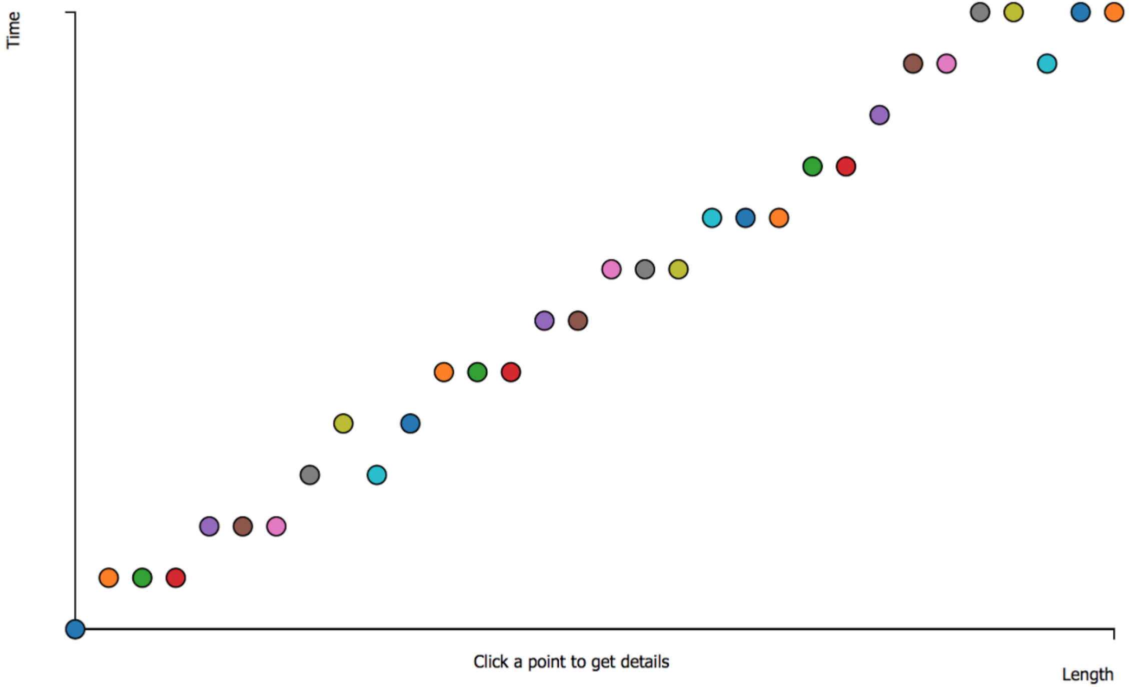
```
// Source/WTF/wtf/text/StringImpl.h

enum CreateEmptyUniqueTag { CreateEmptyUnique };
    StringImpl(CreateEmptyUniqueTag)
        : m_refCount(s_refCountIncrement)
        , m_length(0)
        , m_data16(reinterpret_cast<const UChar*>(1))
    {
        ASSERT(m_data16);

        unsigned hash = static_cast<uint32_t>(reinterpret_cast<uintptr_t>(this));
        WTFLogAlways("Address : 0x%08x\n", hash);
        hash    s_flagCount;
        if (!hash)
            hash = 1 << s_flagCount;
        m_hashAndFlags = hash | BufferInternal;

        STRING_STATS_ADD_16BIT_STRING(m_length);
    }
```

DEMO



- Reported and patched in WebKit

- Related work
  - DoS attacks on hash tables [7,8]
  - Timing attacks on hash tables (Firefox) [9]

Countermeasures

- Non-deterministic hashing for controllable objects
    - Universal Hashing

- Simply not using addresses
    - Random values
        - Possible collisions ?
    - XOR masking
        - Two-time pads?

```
// php-src/ext/spl/php_spl.c

PHPAPI void php_spl_object_hash(zval *obj, char *result TSRMLS_DC) /* {{{*/
{
        intptr_t hash_handle, hash_handlers;
        char *hex;

        if (!SPL_G(hash_mask_init)) {
                if (!BG(mt_rand_is_seeded)) {
                        php_mt_srand(GENERATE_SEED() TSRMLS_CC);
                }

                SPL_G(hash_mask_handle)   = (intptr_t)(php_mt_rand(TSRMLS_C) >> 1);
                SPL_G(hash_mask_handlers) = (intptr_t)(php_mt_rand(TSRMLS_C) >> 1);
                SPL_G(hash_mask_init) = 1;
        }

        hash_handle   = SPL_G(hash_mask_handle)^(intptr_t)Z_OBJ_HANDLE_P(obj);
        hash_handlers = SPL_G(hash_mask_handlers)^(intptr_t)Z_OBJ_HT_P(obj);

        spprintf(&hex, 32, "%016lx%016lx", hash_handle, hash_handlers);

        strlcpy(result, hex, 33);
        efree(hex);
}
```

Random mask init

Two-time pads!

History of ASLR adoption in Android

- Why ASLR on Android?
  - Prevent exploitations of native code in apps

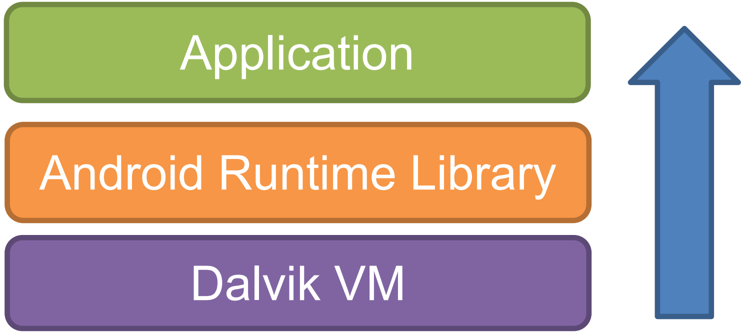
- Adopted incrementally
  - Performance concerns on early Android devices (enabling PIE → load latency / memory overheads)
  - Android 4.1 implemented **full ASLR enforcements**

(actual) ASLR enforcements in Android related to performance prioritized design

Performance Prioritized Designs of Android

- Multi-layered architectures
  - Android Applications run in a Dalvik VM
  - with additional runtime libraries

➔ Slow app launch time

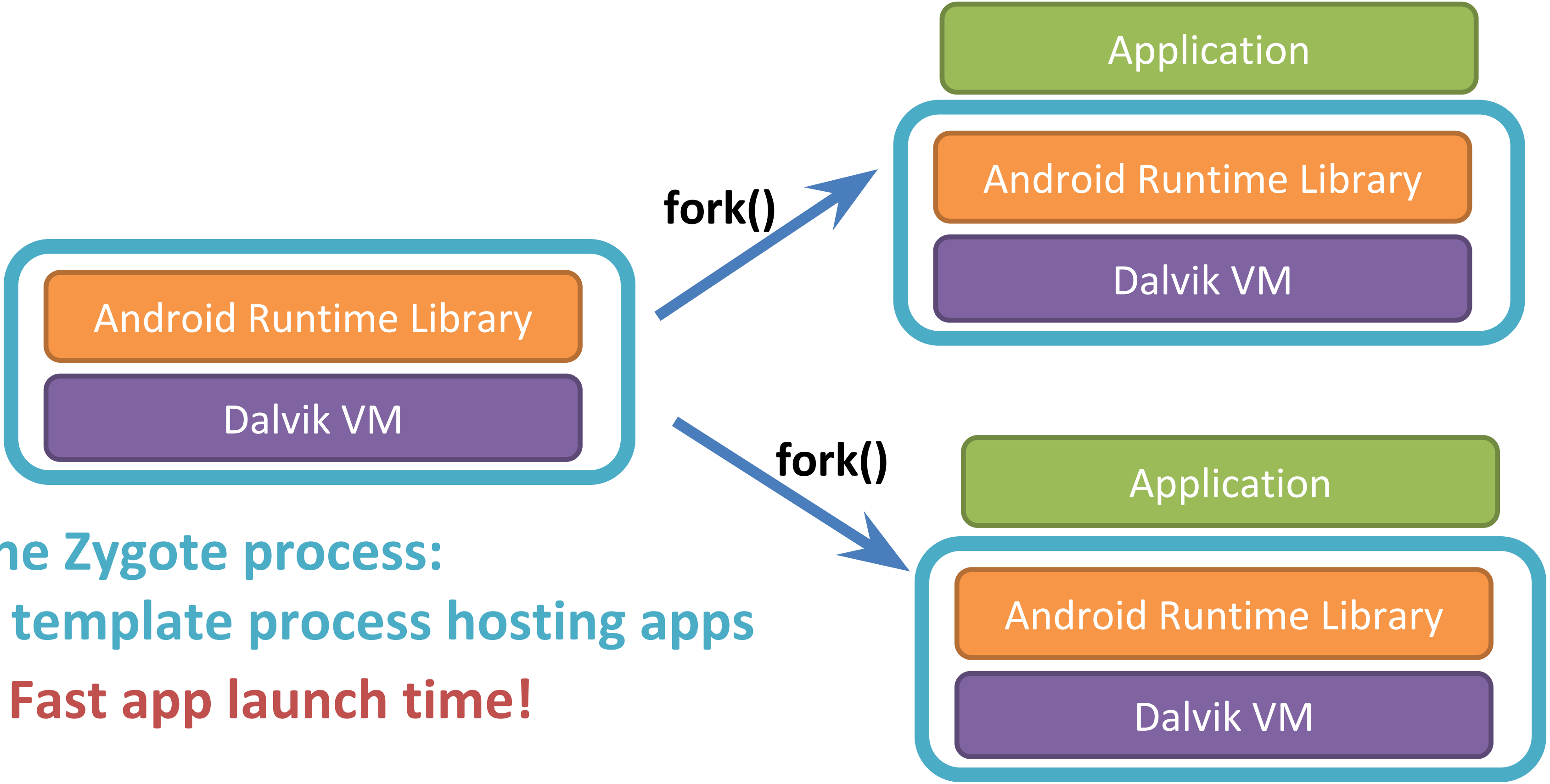


Application

Android Runtime Library

Dalvik VM

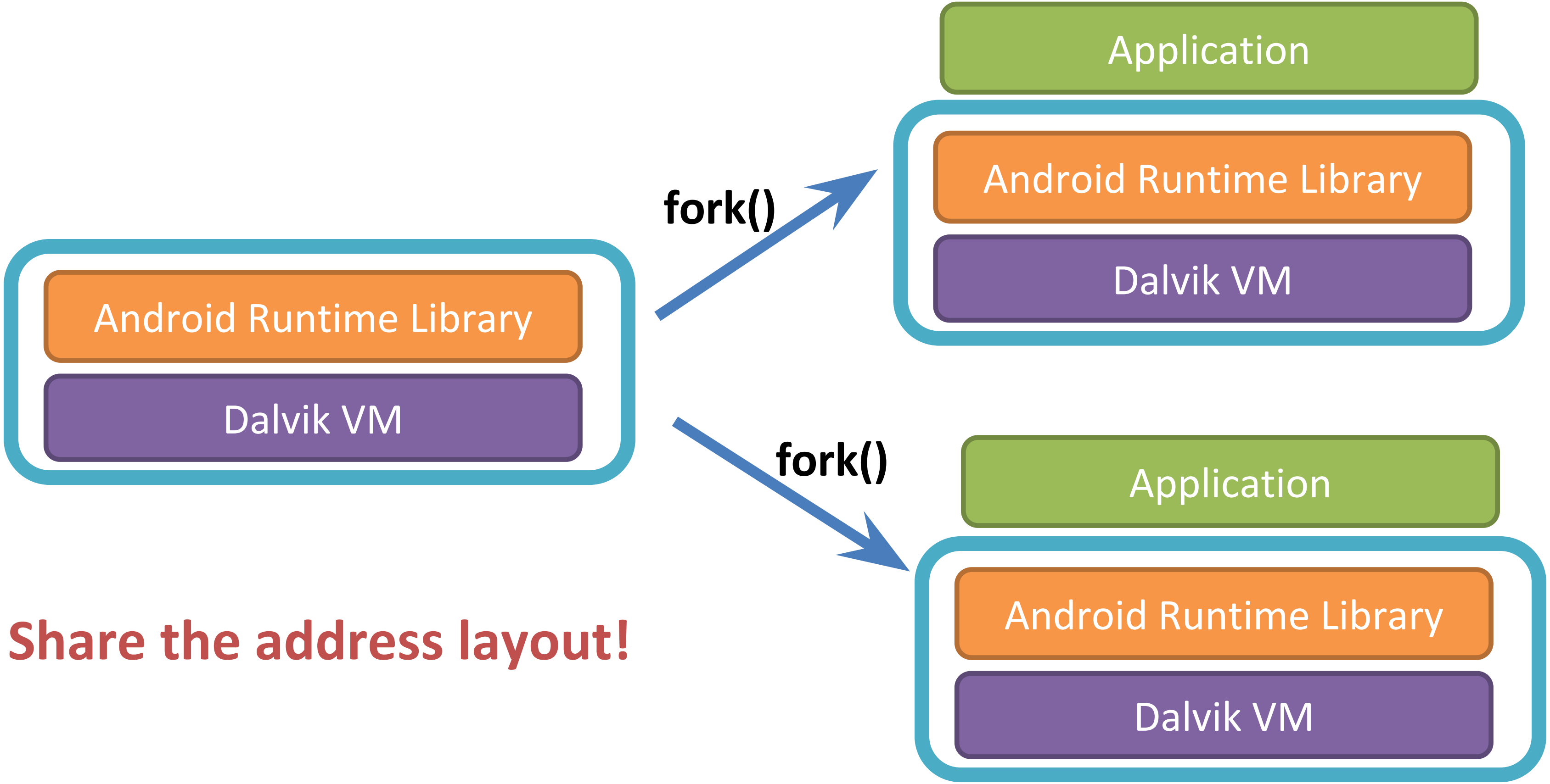
Zygote: the process creation module



the Zygote process:
a template process hosting apps

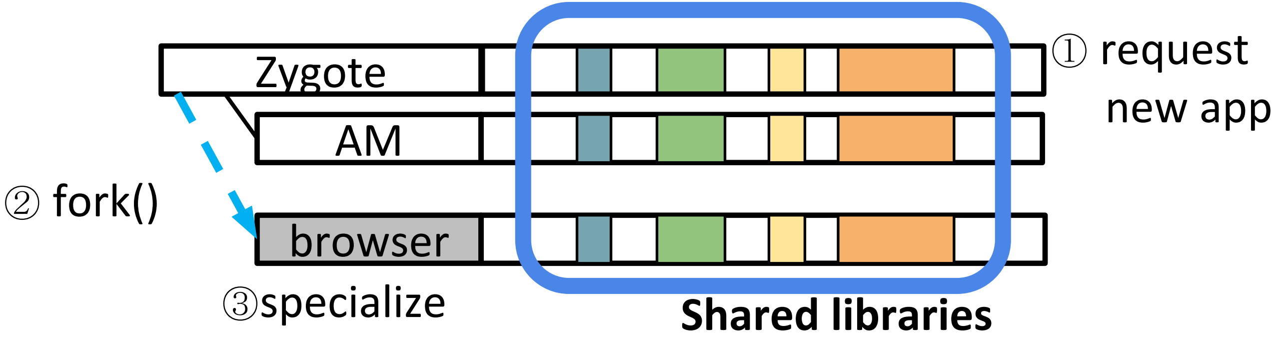
Fast app launch time!

Zygote: the process creation module



Share the address layout!

Zygote weakens ASLR effectiveness



- All apps have the same memory layouts for shared libraries loaded by the Zygote process
- **Weakens Android ASLR security**

Attacking the ASLR weakness

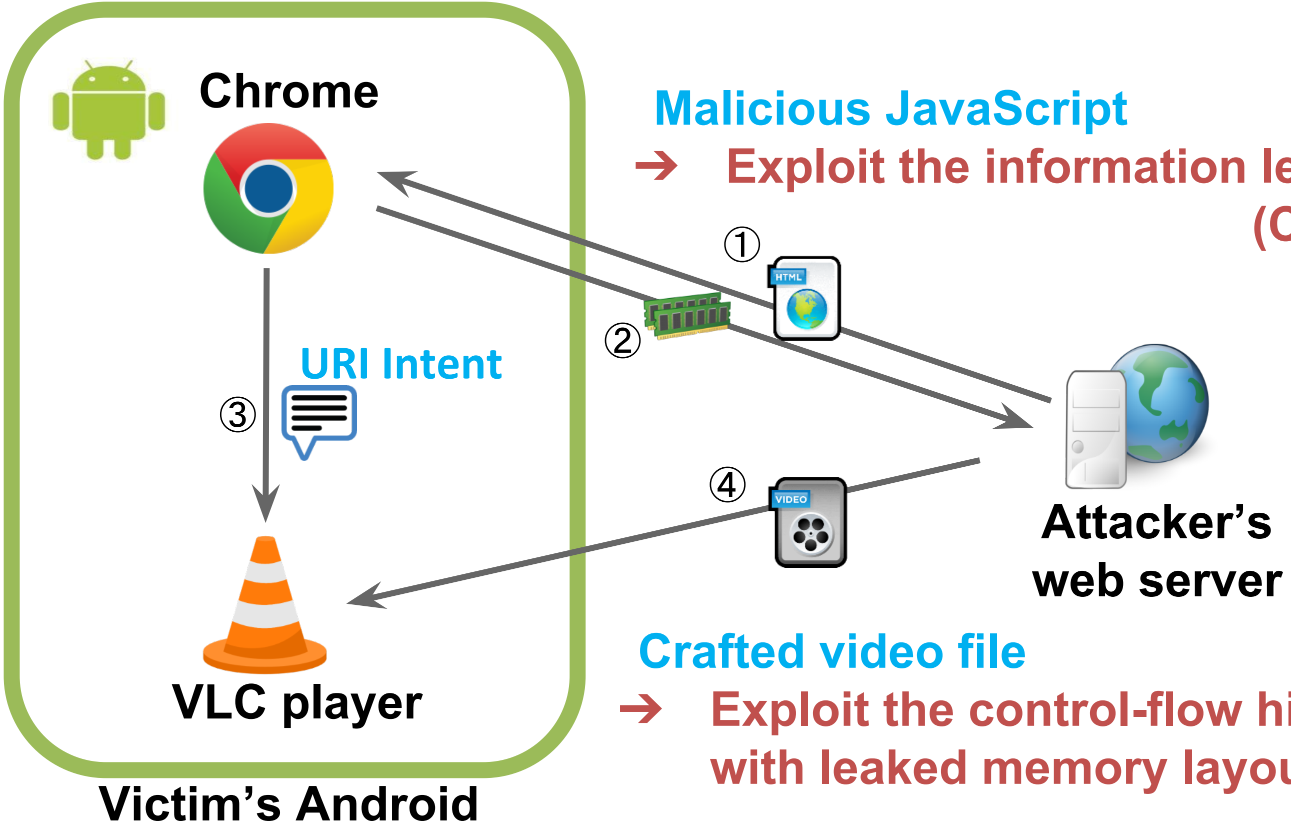
- Fully working exploits (with an ideal ASLR) must
  - Exploit an **Information leak** vulnerability
  - Exploit a **control-flow hijack** vulnerability

  ➔ should be achieved in **the same app**!

Attacking the ASLR weakness

- Zygote’s ASLR weakness allows for
  - **Remote Coordinated Attacks**
    - Information leak in Chrome + control-flow hijack in VLC
    - Reduce the vulnerability searching spaces
  - **Local Trojan Attacks**
    - Obtain the memory layout by having the trojan app installed

Remote Coordinated Attack



Chrome

① ②

URI Intent

③

VLC player

Victim's Android

④

Attacker's
web server

Malicious JavaScript

→ Exploit the information leak vulnerability (CVE-2013-0912)

Crafted video file

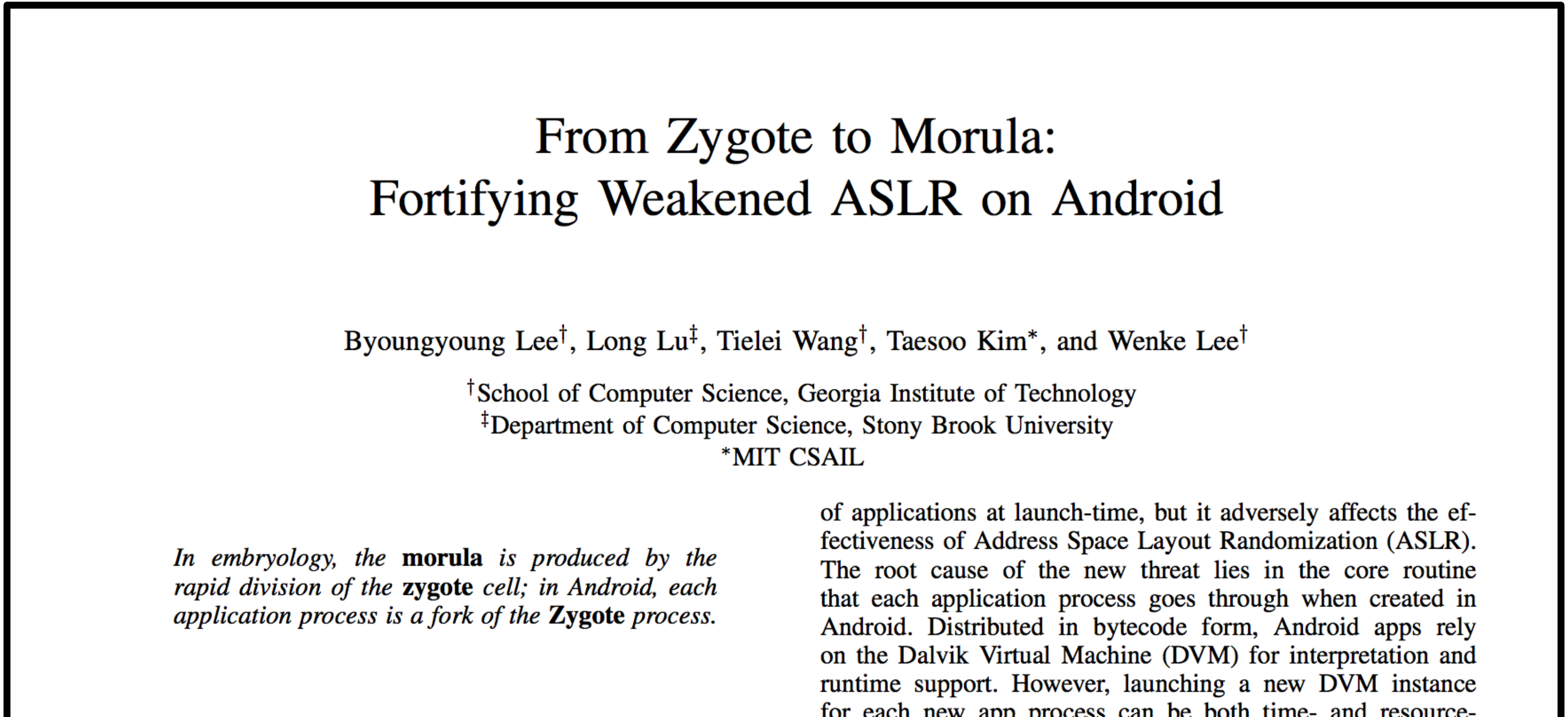
→ Exploit the control-flow hijack vulnerability with leaked memory layout information

Local Trojan Attack

- Zero permission trojan app
  - Asks for (almost) no permissions
  - Scans memory spaces using app native code
  - Layout information can be further exported

- Once a trojan app is installed, ASLR can be easily bypassed

Countermeasures



From Zygote to Morula:
Fortifying Weakened ASLR on Android

Byoungyoung Lee[†], Long Lu[‡], Tielei Wang[†], Taesoo Kim[*], and Wenke Lee[†]

[†]School of Computer Science, Georgia Institute of Technology
[‡]Department of Computer Science, Stony Brook University
[*]MIT CSAIL

In embryology, the **morula** *is produced by the rapid division of the* **zygote** *cell; in Android, each application process is a fork of the* **Zygote** *process.*

of applications at launch-time, but it adversely affects the effectiveness of Address Space Layout Randomization (ASLR). The root cause of the new threat lies in the core routine that each application process goes through when created in Android. Distributed in bytecode form, Android apps rely on the Dalvik Virtual Machine (DVM) for interpretation and runtime support. However, launching a new DVM instance for each new app process can be both time- and resource-

References

1. S. Forrest, A. Somayaji, and D. Ackley. Building Diverse Computer Systems. In Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI) (HOTOS '97). 1997.
2. Ken Johnson, MaF Miller. Exploit Mitigation Improvements in Windows 8. Black hat USA, 2012.
3. Yang Yu. DEP/ASLR bypass without ROP/JIT. CanSecWest 2013.
4. Andrea Bittau, et al. Hacking Blind. IEEE S&P. 2014.
5. Fermin J. Serna. The info leak era on software exploitation. Blackhat USA, 2012.
6. Xiaobo Chen. ASLR Bypass Apocalypse in Recent Zero-Day Exploits. http://www.fireeye.com/blog/technical/cyber-exploits/2013/10/aslr-bypass-apocalypse-in-lately-zero-day-exploits.html
7. Scott A. Crosby, and Dan S. Wallach, Denial of Service via Algorithmic Complexity Attacks. USENIX Security
8. Alexander Klink, and Julian Walde, Efficient Denial of Service Attacks on Web Application Platforms. CCC 2011
9. pakt, and Dion Blazakis, Leaking addresses with vulnerabilities that can’t read good. Summercon 2013
10. A. Bittau, A. Belay, A. Mashtizadeh, and D. Mazières, D. Boneh: Hacking Blind. Oakland 2014

Thank you! :)

Questions?