司南: 定位二进制代码中的不安全密钥

李卷孺

上海交通大学 蜚语（G.O.S.S.I.P）软件安全小组

网络安全研究国际论坛（Inforsec）

北京 2019年1月18日

密码安全攻防——安全研究的新热门

Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2 @ CCS 2017

已有研究工作

- 移动平台的密码学误用检测
  - **CryptoLint** (Android) @ *CCS 2013*
  - **iCryptoTracer** (iOS) @ *NSS 2014*
  - **NativeSpeaker** (Android) @ *Inscrypt 2017*
- 密码算法的自动化检测
  - **Aligot** @ *CCS 2012*
  - **CipherXRay** @ *TDSC 2012*
  - **CryptoHunt** @ *Oakland 2017*
- 密码参数的提取
  - **ReFormat** @ *ESORICS 2009*
  - **Dispatcher** @ *CCS 2009*
  - **MovieStealer** @ *Usenix Security 2013*

密钥（Crypto Keys）——核心机密

- Kerckhoffs's principle
  - ***一切秘密寓于密钥之中***
  - *a.k.a* ***the enemy knows the system***（香农）
- 近年来，涌现了各类针对密钥的攻击

Lest we remember @ 2009

Heartbleed @ 2014

Foreshadow @ 2018

问题：如何实现通用的密钥分析

```
1  uint8_t Key[16];
2  uint8_t Data[256] = {0};
3
4  void keygen(uint8_t * key, size_t len)
5  {
6          uint8_t seed[4];
7          for ( size_t i = 0; i < 4; ++i )
8                  seed[i] = rand() & 0xff;
9          for ( size_t i = 0; i < len; ++i )
10                 key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15         for ( size_t i = 0; i < len; ++i )
16                 buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21         keygen(Key, 16);
22         encrypt(Data, 256);
23 }
```

问题：如何实现通用的密钥分析

```
1  uint8_t Key[16];
2  uint8_t Data[256] = {0};
3
4  void keygen(uint8_t * key, size_t len)
5  {
6          uint8_t seed[4];
7          for ( size_t i = 0; i < 4; ++i )
8                  seed[i] = rand() & 0xff;
9          for ( size_t i = 0; i < len; ++i )
10                 key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15         for ( size_t i = 0; i < len; ++i )
16                 buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21         keygen(Key, 16);
22         encrypt(Data, 256);
23 }
```

密钥生成缺乏随机性！

问题：如何实现通用的密钥分析

```
1  uint8_t Key[16];
2  uint8_t Data[256] = {0};
3
4  void keygen(uint8_t * key, size_t len)
5  {
6         uint8_t seed[4];
7         for ( size_t i = 0; i < 4; ++i )
8               seed[i] = rand() & 0xff;
9         for ( size_t i = 0; i < len; ++i )
10              key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15        for ( size_t i = 0; i < len; ++i )
16              buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21        keygen(Key, 16);
22        encrypt(Data, 256);
23 }
```

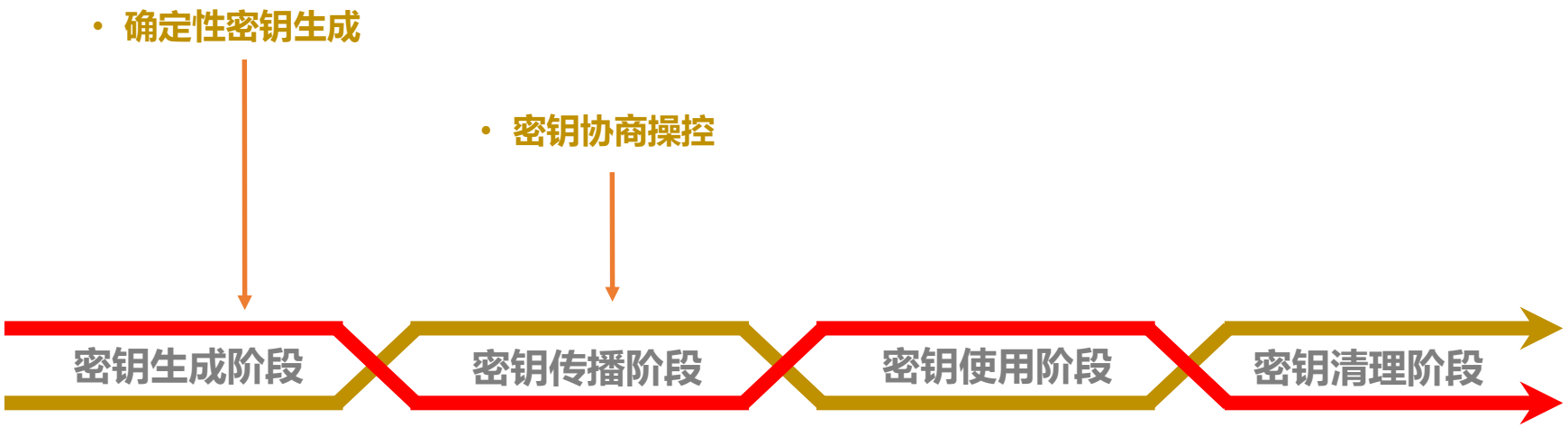
密钥内存并未正确清理

密码算法无关的不安全密钥使用

- 确定性密钥生成

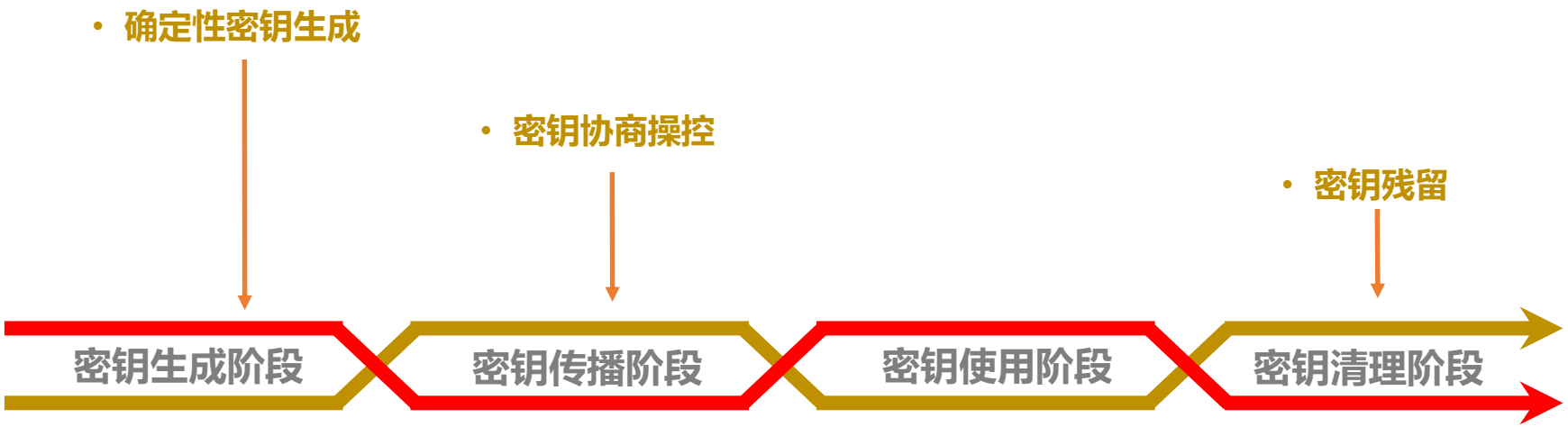
密钥的整个生命周期

密码算法无关的不安全密钥使用



密钥的整个生命周期

密码算法无关的不安全密钥使用



密钥的整个生命周期

如何开展程序的密码安全分析

```
1  uint8_t Key[16];
2  uint8_t Data[256] = {0};
3
4  void keygen(uint8_t * key, size_t len)
5  {
6          uint8_t seed[4];
7          for ( size_t i = 0; i < 4; ++i )
8                  seed[i] = rand() & 0xff;
9          for ( size_t i = 0; i < len; ++i )
10                 key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15         for ( size_t i = 0; i < len; ++i )
16                 buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21         keygen(Key, 16);
22         encrypt(Data, 256);
23 }
```

1. 定位密码函数

如何开展程序的密码安全分析

2. 分析内存块的语义信息，以便找到密钥块

```
1  uint8_t Key[16];
2  uint8_t Data[256] = {0};
3
4  void keygen(uint8_t * key, size_t len)
5  {
6          uint8_t seed[4];
7          for ( size_t i = 0; i < 4; ++i )
8                  seed[i] = rand() & 0xff;
9          for ( size_t i = 0; i < len; ++i )
10                 key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15         for ( size_t i = 0; i < len; ++i )
16                 buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21         keygen(Key, 16);
22         encrypt(Data, 256);
23 }
```

1. 定位密码函数

如何开展程序的密码安全分析

2. 分析内存块的语义信息，以便找到密钥块

```
1  uint8_t Key[16];
2  uint8_t Data[256] = {0};
3
4  void keygen(uint8_t * key, size_t len)
5  {
6          uint8_t seed[4];
7          for ( size_t i = 0; i < 4; ++i )
8                  seed[i] = rand() & 0xff;
9          for ( size_t i = 0; i < len; ++i )
10                 key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15         for ( size_t i = 0; i < len; ++i )
16                 buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21         keygen(Key, 16);
22         encrypt(Data, 256);
23 }
```

3. 追踪密钥的传播

1. 定位密码函数

如何开展程序的密码安全分析

2. 分析内存块的语义信息，以便找到密钥块

```
1  uint8_t Key[16];
2  uint8_t Data[256] = {0};
3
4  void keygen(uint8_t * key, size_t len)
5  {
6          uint8_t seed[4];
7          for ( size_t i = 0; i < 4; ++i )
8                  seed[i] = rand() & 0xff;
9          for ( size_t i = 0; i < len; ++i )
10                 key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15         for ( size_t i = 0; i < len; ++i )
16                 buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21         keygen(Key, 16);
22         encrypt(Data, 256);
23 }
```

3. 追踪密钥的传播

1. 定位密码函数

4. 检查密钥在程序执行结束时的状态

主要的困难

- **代码和算法的多样性**
  - 私有密码算法：缺乏相关的知识
  - 私有实现的算法：差异性非常大

主要的困难

- 代码和算法的多样性
  - 私有密码算法：缺乏相关的知识
  - 私有实现的算法：差异性非常大

- **代码量导致的分析成本**
  - 海量代码导致的分析时间
  - 如何精确定位密码函数的边界

主要的困难

- 代码和算法的多样性
  - 私有密码算法：缺乏相关的知识
  - 私有实现的算法：差异性非常大
- 代码量导致的分析成本
  - 海量代码导致的分析时间
  - 如何精确定位密码函数的边界
- **密码语义恢复**
  - 如何识别出存放密钥信息的内存区域

解决问题之道

- 不去识别具体的密码算法！
  - **利用动态语义特性，直接去寻找那些和密码操作强相关的代码基本块**

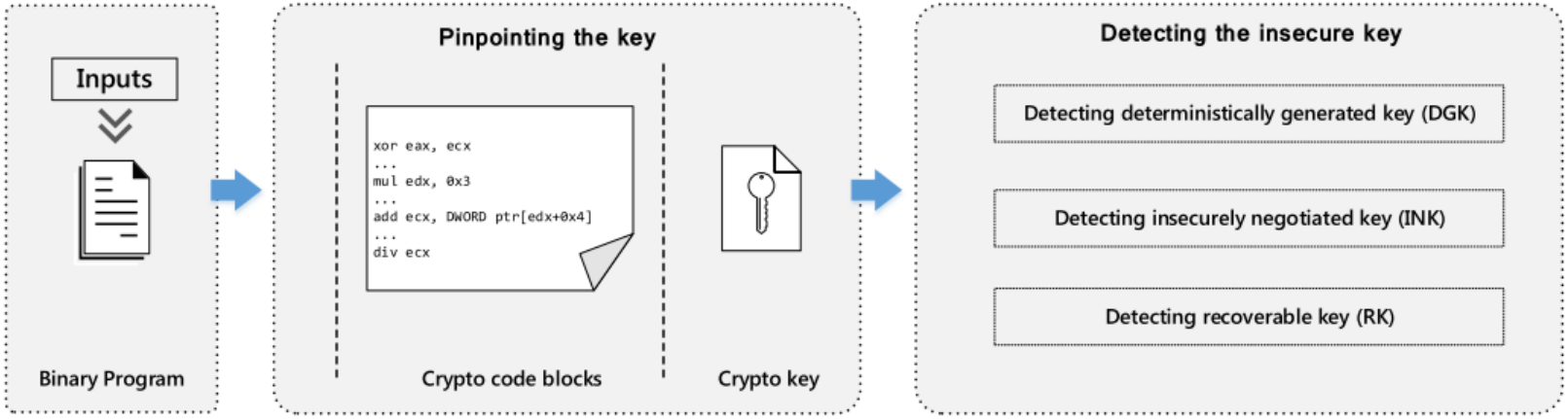
解决问题之道

- 不去识别具体的密码算法！
  - 利用动态语义特性，直接去寻找那些和密码操作强相关的代码基本块
- 不去静态分析二进制代码
  - **通过运行时记录的程序执行信息（traces）来开展动态分析**

解决问题之道

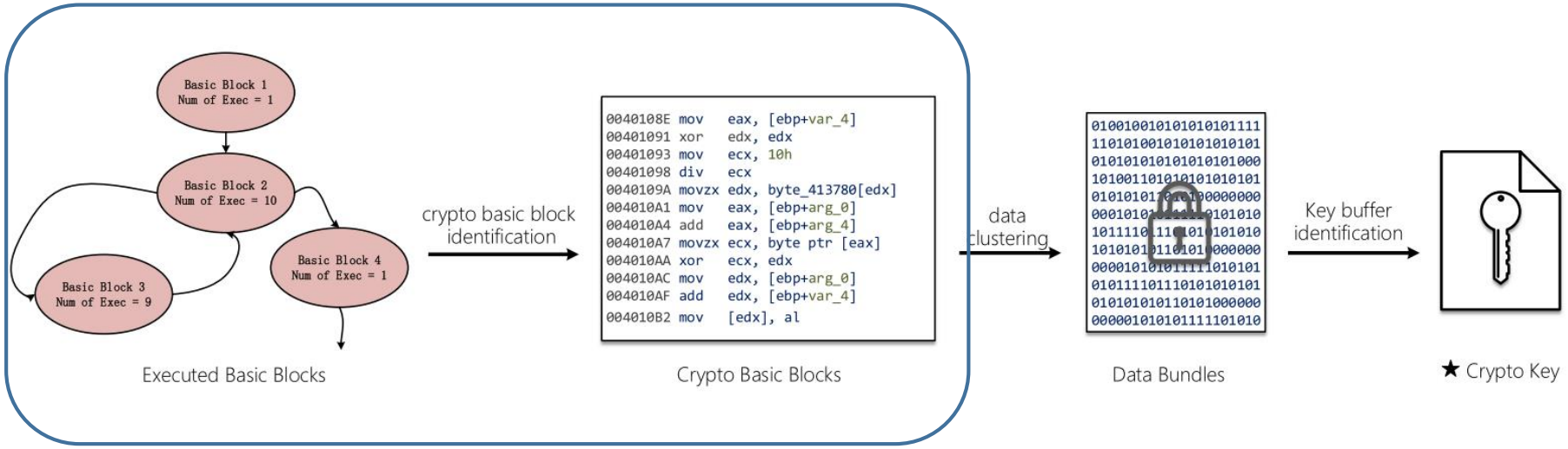
- 不去识别具体的密码算法！
  - 利用动态语义特性，直接去寻找那些和密码操作强相关的代码基本块
- 不去静态分析二进制代码
  - 通过运行时记录的程序执行信息（traces）来开展动态分析
- 不去进行特定的密码误用分析
  - **动态定位密码算法无关的不安全密钥使用**

司南：二进制代码中的密钥安全分析系统



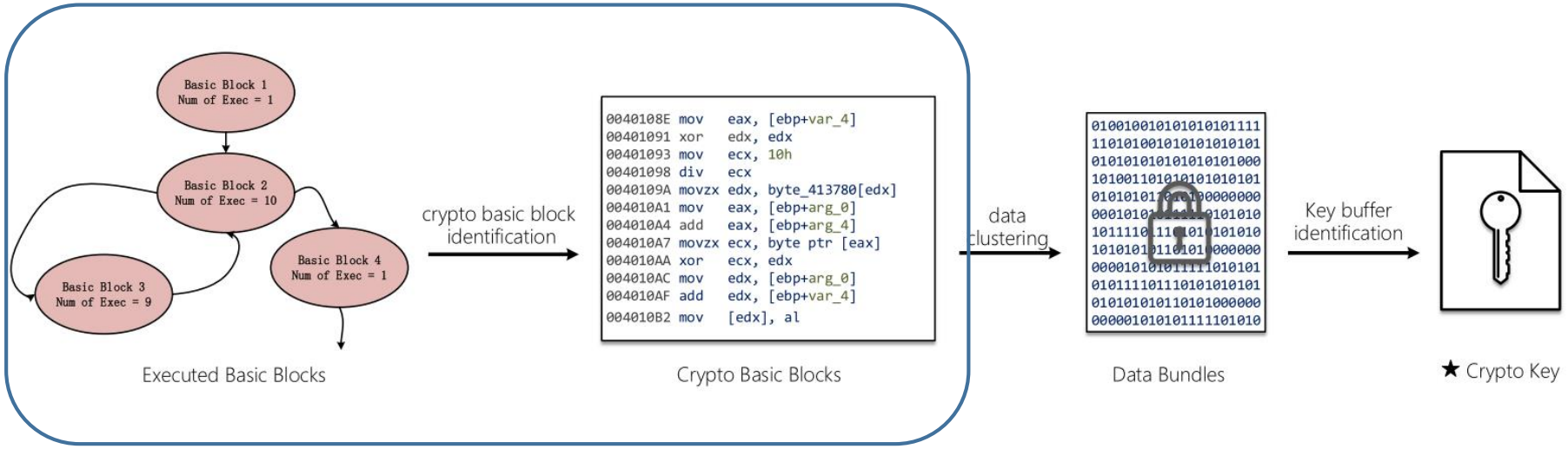
- 基于Intel PIN二进制代码插桩框架
- 支持Windows、Linux和macOS上的x86/64 二进制代码
- 分析过程包括两个阶段：
  - **定位使用的密钥**
  - **检测不安全密钥**

定位使用的密钥



- **第一步：寻找密码基本块**
  - 具备一定比例的算术运算指令或者特定密码指令（AES-NI）
  - 执行次数随着不同规模的输入/输出而（线性）变化
  - 操作的数据具有极高的随机性

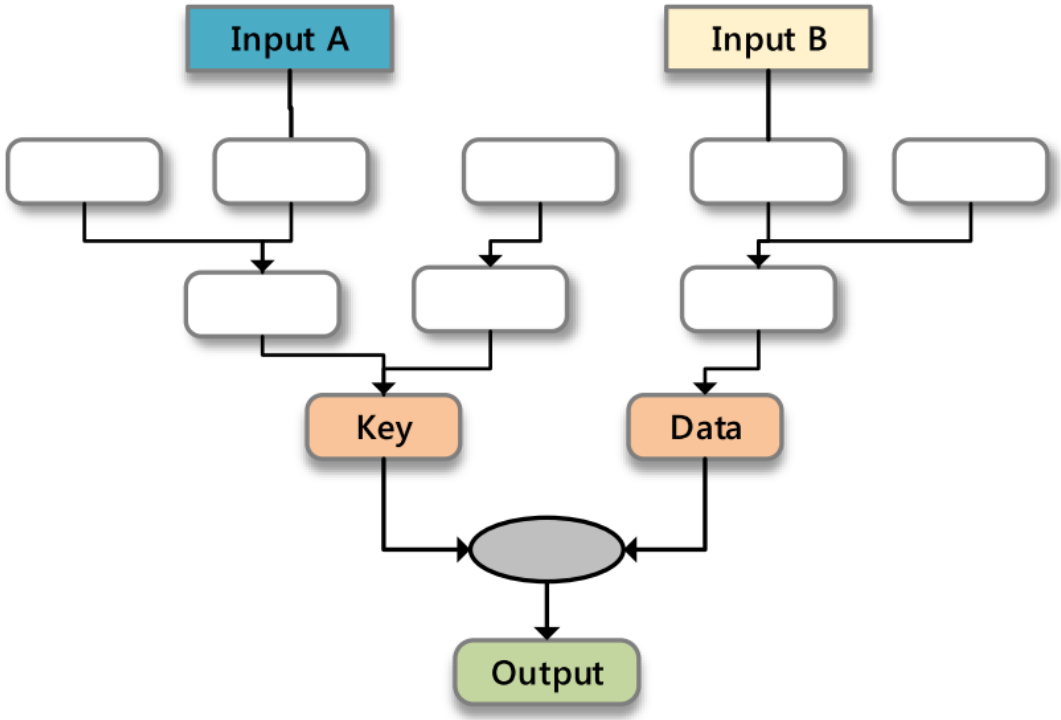
定位使用的密钥



- 第一步：寻找密码基本块
  - 具备一定比例的算术运算指令或者特定密码指令（AES-NI）
  - 执行次数随着不同规模的输入/输出而（线性）变化
  - 操作的数据具有极高的随机性
- **第二步：寻找密钥内存块**
  - 根据不同内存块的初始化和使用情况聚类
  - 根据不同执行情况下的变化和不变情况分类

检测不安全的密钥

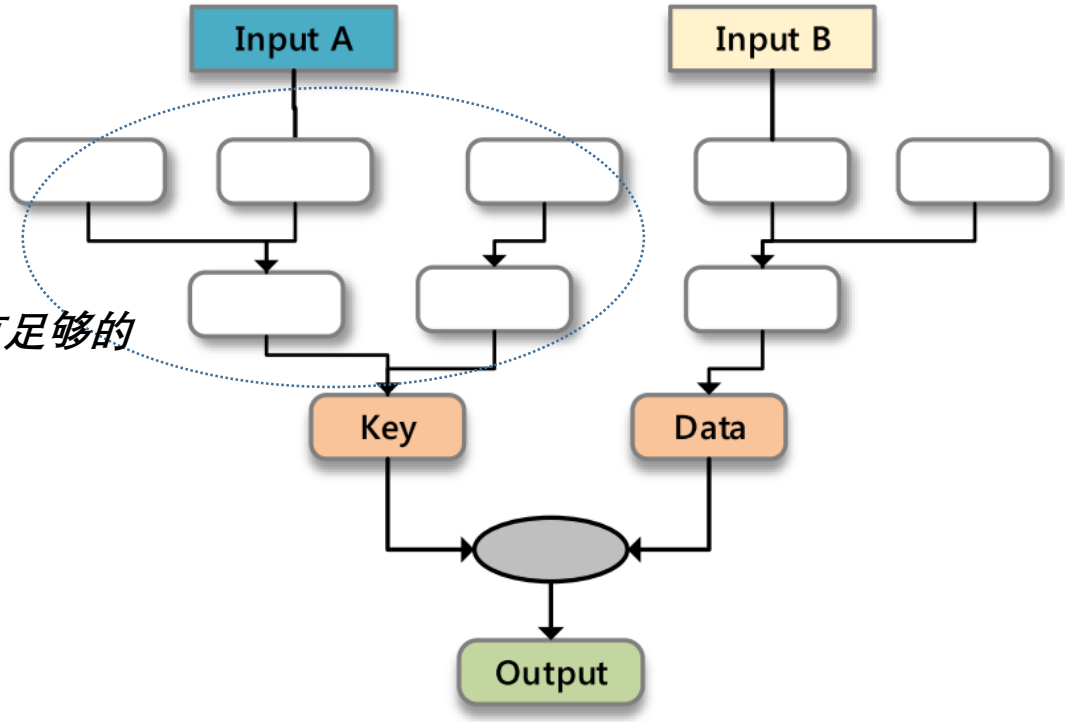
- 提出了一种基于粗粒度（函数级别）动态污点分析的不安全密钥检测方法



检测不安全的密钥

- 提出了一种基于粗粒度（函数级别）动态污点分析的不安全密钥检测方法

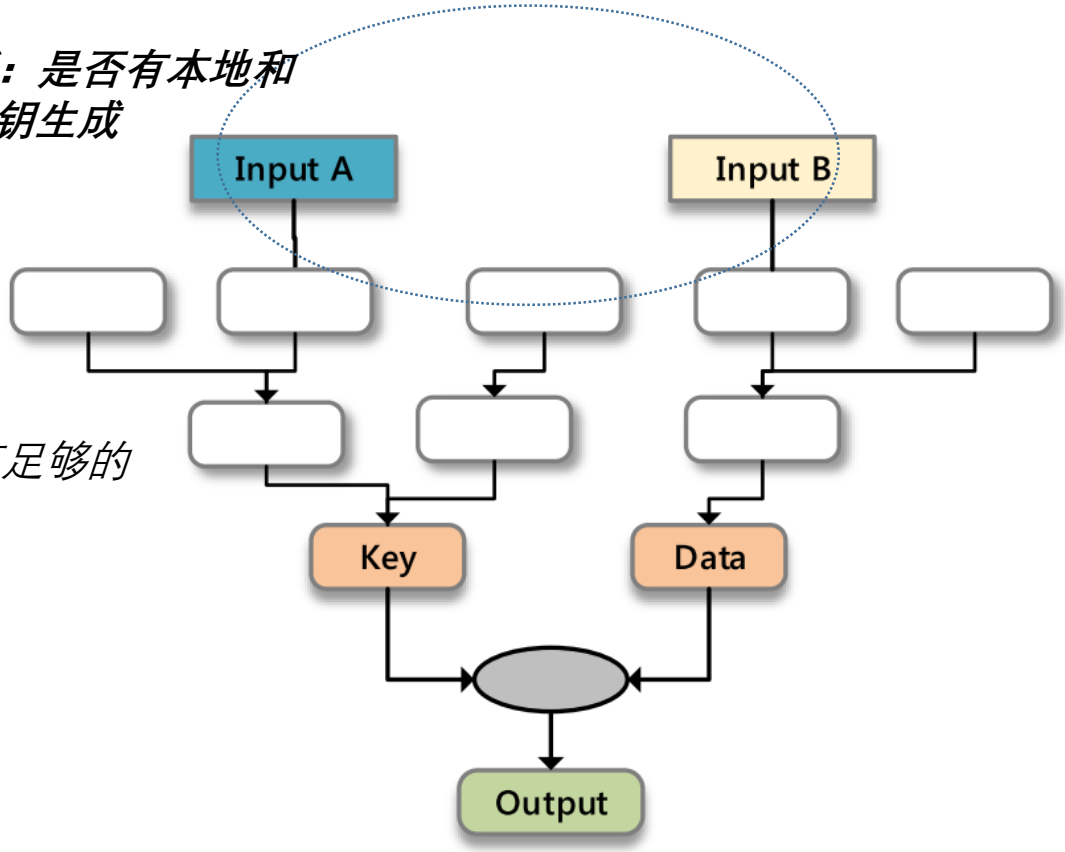
1. 传播过程分析：是否有足够的随机性传播到密钥



检测不安全的密钥

- 提出了一种基于粗粒度（函数级别）动态污点分析的不安全密钥检测方法

2. 会话密钥分析：是否有本地和远程输入参与密钥生成



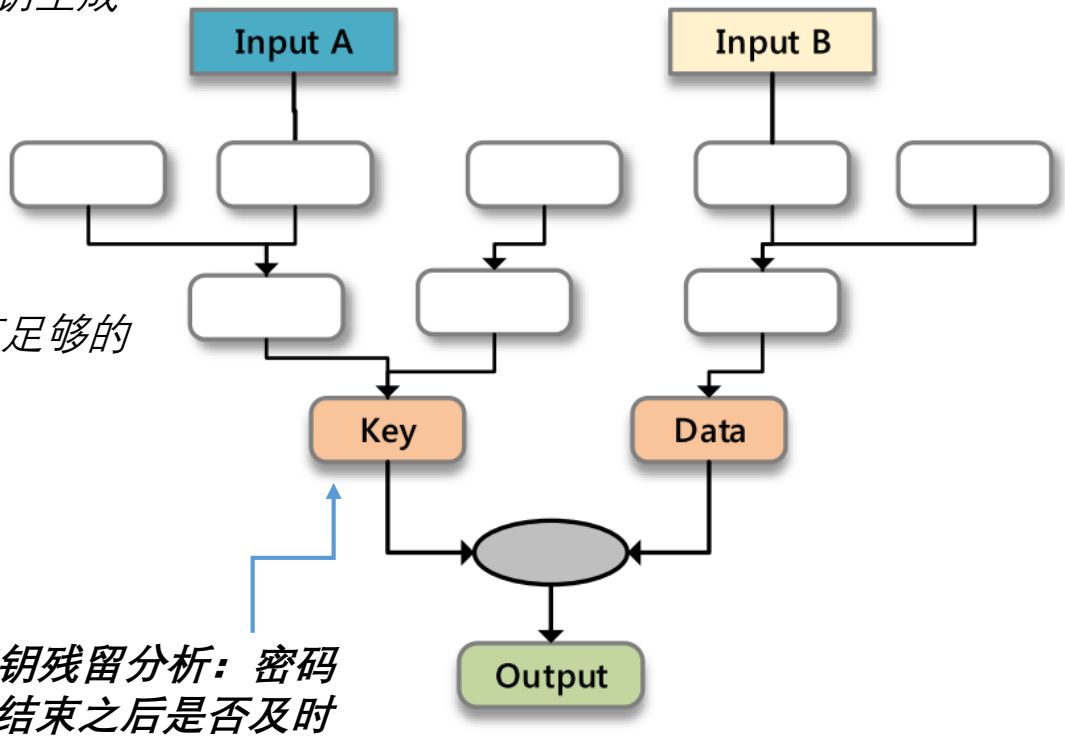
1. 传播过程分析：是否有足够的随机性传播到密钥

检测不安全的密钥

- 提出了一种基于粗粒度（函数级别）动态污点分析的不安全密钥检测方法

2. 会话密钥分析：是否有本地和远程输入参与密钥生成

1. 传播过程分析：是否有足够的随机性传播到密钥



3. 密钥残留分析：密码操作结束之后是否及时清理密钥信息

实验对象

- 主流密码学算法库
  - 选择了10个不同的算法库，为其编写测试用例以覆盖三类算法（AES、RSA、ECDSA）

OpenSSL Cryptography and SSL/TLS Toolkit

Crypto++

ARM mbed

wolfSSL

libsodium

- 包含密码算法的工具
  - 选择了15个不同的工具，其中包含了不同算法（包括私有密码算法和标准算法私有实现）

7ZIP

密钥识别情况（1）

Target	Algorithm	B1	B2	B3	N	S	IL
Botan	AES-256	53	13	7	1	240	32
	RSA-2048	1180	569	162	6	1024	256
	ECDSA	958	921	300	2	224	128
Crypto++	AES-256	1281	26	5	1	240	32
	RSA-2048	1949	924	214	6	896	256
	ECDSA	1916	1425	305	8	288	64
Libgcrypt	AES-256	126	25	3	1	240	32
	RSA-2048	565	463	153	6	896	896
	ECDSA	340	322	49	10	320	96
LibSodium	AES NI-256	7	4	4	1	240	32
	Ed25519	690	686	171	8	288	256
LibTomcrypt	AES-256	60	43	4	1	240	32
	RSA-2048	404	385	69	7	1152	1152
	ECDSA	330	274	72	4	128	97
Nettle	AES-256	38	13	3	1	240	32
	RSA-2048	411	87	61	6	1152	896
	ECDSA	186	92	39	8	288	32
mbedTLS	AES-256	44	40	13	1	240	32
	RSA-2048	154	138	39	12	1664	256
	ECDSA	255	245	47	9	384	64
OpenSSL	AES-256	58	10	4	1	240	32
	RSA-2048	210	175	41	10	1552	640
	ECDSA	188	143	17	6	192	50
WolfSSL	AES-256	50	36	4	1	240	32
	RSA-2048	295	235	36	7	1152	1152
	ECDSA	277	202	27	5	160	32

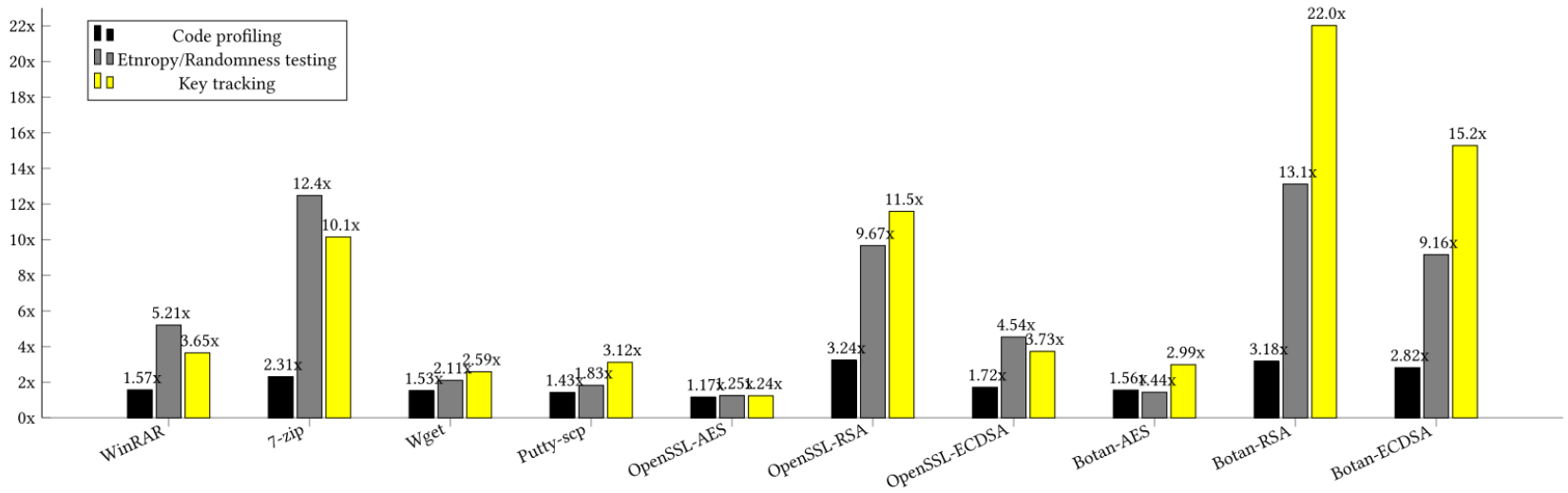
- **B1**: 经过算术指令比例过滤后剩余的基本块数量；
- **B2**: 在B1中，经过输入相关性分析后剩余的基本块数量；
-
- **B3**: 在B2中，经过数据随机性分析后剩余的基本块数量（也就是最终认定的密钥基本块）；
- **N**: 在密码基本块中找到的密钥内存块数量；
- **S**: 密钥内存块的大小总和（字节）；
- **IL**: 密钥内存块中和输入相关的信息量（字节）

密钥识别情况（2）

Target	Algorithm	B1	B2	B3	N	S	IL
7-zip	AES NI-256	2	2	2	1	240	32
Ccrypt	AES-256	44	5	1	1	240	32
Cryptcat	Twofish	54	14	7	1	160	varied
Cryptochief	Proprietary *	23	12	1	1	8	3
Enpass	AES NI-256	8	3	3	1	240	32
Imagine	DSA-1024 *	241	72	12	5	464	928
IpMsg	AES-256	168	12	4	1	240	32
Keepass	AES-256	481	118	19	1	240	32
MuPDF	AES-128	262	46	4	1	176	16
PSCP	AES-256	195	9	5	1	240	32
Sage	ChaCha20 *	31	17	2	1	256	32
UltraSurf	RC4 *	191	79	6	1	1024	16
WannaCry	AES-128 *	26	12	3	1	352	16
Wget	AES-256	268	22	3	1	240	32
WinRAR	AES-128 *	181	58	3	1	176	32
	AES-256 *	214	51	3	1	240	48

- 对于测试的10个密码学算法库和15个包含密码算法的工具软件，司南成功地定位了所有使用的密码算法，并找到了密钥内存区域；
- 即使是**私有的密码算法（cryptochief中使用的私有流密码算法）**以及**标准算法的私有实现（勒索软件中的AES实现）**，司南也能准确定位；
- **密钥内存区域的布局差异，并不影响我们的动态分析结果的准确性！**

分析开销



司南中包含的三个 pintools 同PIN framework的null pintool的开销比较

检测到的不安全密钥使用情况

Target	DGK	INK	RK			
			NMZ	MMZ	RKPS	RKPH
Botan	-	-	-	-	-	-
Crypto++	-	-	-	-	-	-
Libgcrypt	-	-	-	✓	-	-
LibSodium	-	-	✓	-	-	-
LibTomcrypt	-	-	✓	-	-	-
Nettle	-	-	✓	-	-	-
GnuTLS	-	-	-	✓	-	-
mbedTLS	-	-	-	✓	-	-
OpenSSL	-	-	-	✓	-	-
WolfSSL	-	-	✓	-	-	-
7-zip	-	-	-	-	✓	-
Ccrypt	-	-	-	-	-	✓
Cryptcat	-	-	-	-	-	✓
Cryptochief	✓	-	-	-	-	✓
Enpass	-	-	-	-	✓	-
Imagine	✓	-	-	-	-	-
IpMsg	-	✓	-	-	✓	-
Keepass	-	-	-	-	✓	-
MuPDF	-	-	-	-	✓	-
PSCP	-	-	-	-	-	-
Sage	-	-	-	-	✓	-
UltraSurf	-	✓	-	-	✓	-
WannaCry	-	-	-	-	✓	-
Wget	-	-	-	-	✓	-
WinRAR	-	-	-	-	-	✓

- **NMZ:** 没有密钥清理
- **MMZ**: 依赖程序员手工清理
- **RKPS**: 在栈上存在密钥残留
- **RKPH**: 在堆上存在密钥残留

- 在25个测试对象中，有**22**个程序不安全地使用了密钥!
- 甚至那些开发时间超过10年的密码库和工具软件都会犯下密钥残留的基本错误
- 私有密码方案中容易出现密钥随机性不足的问题
- 无证书的混合加密体系通讯过程中，密钥协商操控比较普遍

Case Study: Imagine中的不安全密钥

Imagine (一个韩国的图像浏览软件）：使用了DSA签名算法来验证注册

DSA Signing

$$r = g^k \mod p \mod q \quad (1)$$

$$s = k^{-1}(H(m) + x \cdot r) \mod q \quad (2)$$

DSA Verifying

$$w = s^{-1} \mod q \quad (3)$$

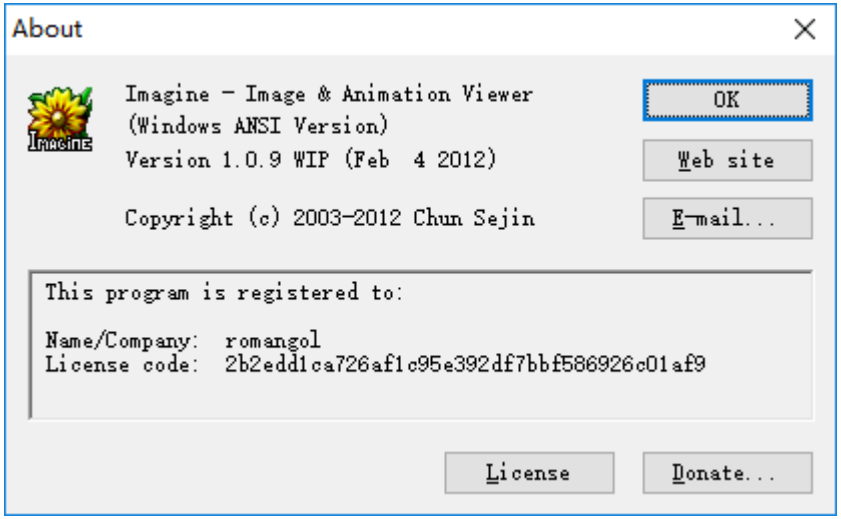
$$u_1 = H(m) \cdot w \mod q \quad (4)$$

$$u_2 = r \cdot w \mod q \quad (5)$$

$$v = (g^{u_1} \cdot y^{u_2} \mod p) \mod q \quad (6)$$

“a hard-coded ***k*** leads an attacker to compute the private key ***x*** with a legal pair of signature (r, s), and thus to forge the signature”

$$x = r^{-1}(k \cdot s - H(m)) \mod q$$



Case Study: Libsodium中的密钥残留

Libsodium's patch against insecurely used AES round keys:
https://github.com/jedisct1/libsodium/commit/28cac20a7bedd2ff35379874e63a33f6168ba31a

Symbolically clear the round keys after aes256gcm_(en|de)crypt()

Browse files

Fixes #617

bench-1.0.16 + stable

jedisct1 committed on 6 Nov 2017 — 1 parent 7b05b7d — commit 28cac20a7bedd2ff35379874e63a33f6168ba31a

```
861       -   return crypto_aead_aes256gcm_encrypt_afternm
      862 +   ret = crypto_aead_aes256gcm_encrypt_afternm
862   863         (c, clen_p, m, mlen, ad, adlen, nsec, npub,
863   864             (const crypto_aead_aes256gcm_state *) &ctx);
      865 +   sodium_memzero(ctx, sizeof ctx);
      866 +
      867 +   return ret;
```

我们通报了各个软件的开发人员（比较懒，并没有去申请CVE），有些很快（虽然挣扎了一下）就回应并修复了漏洞

然而，也有很多软件开发人员选择了沉默……

总结

- **司南**：一个动态安全分析系统，用于检测二进制代码中的不安全密钥使用情况
- **司南**实现了三类**密码算法无关**的不安全密钥检测：确定性密钥生成、密钥协商操控、密钥残留
- **司南**在流行的密码算法库（例如Libsodium ）和流行的工具软件（例如Keepass ）中均发现了不安全密钥使用的情况！