

DroidKex: Fast Extraction of Ephemeral TLS Keys from the Memory of Android Apps

Benjamin Taubmann

Universität Passau
Assistant Professorship of Security in Information Systems

July 17, 2018



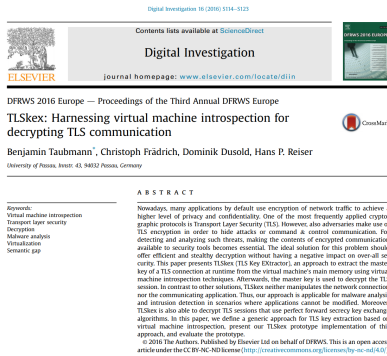
Goal: Decrypt TLS encrypted communication channels for Android applications

TLS - Perfect Forward Secrecy (PFS)

- ▶ Each connection has its own session key
- ▶ Key is negotiated via DH/ECDH

How to decrypt/analyze the communication?

- ▶ **With private key:** Does not work with PFS, but with old RSA-based key exchange
- ▶ **Proxy:** Manipulates the traffic, is detectable, does not work with certificate pinning



TLSkex

- ▶ Monitor network traffic of a virtual machine
- ▶ Take snapshot of main memory after the key negotiation
- ▶ Extract session keys (master secret)
- ▶ **Requirement:** Access to main memory of communicating system is given, e.g., to the memory of a VM

Performance: Sequentially searching for session keys is slow

¹Taubmann, Benjamin et al. "TLSkex: Harnessing virtual machine introspection for decrypting TLS communication." In: *DFRWS EU. 2016*.

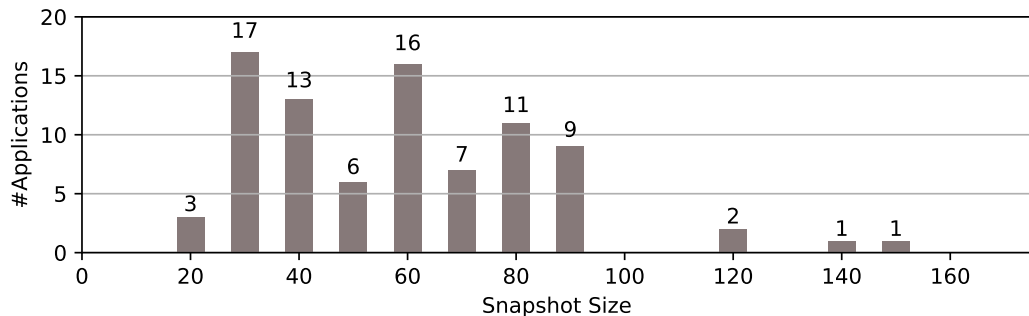


Figure: Histogram: Number of applications with the same snapshot memory size (Size in MiB)

All measurements are executed on a Nexus 5x with Android 6.0.1

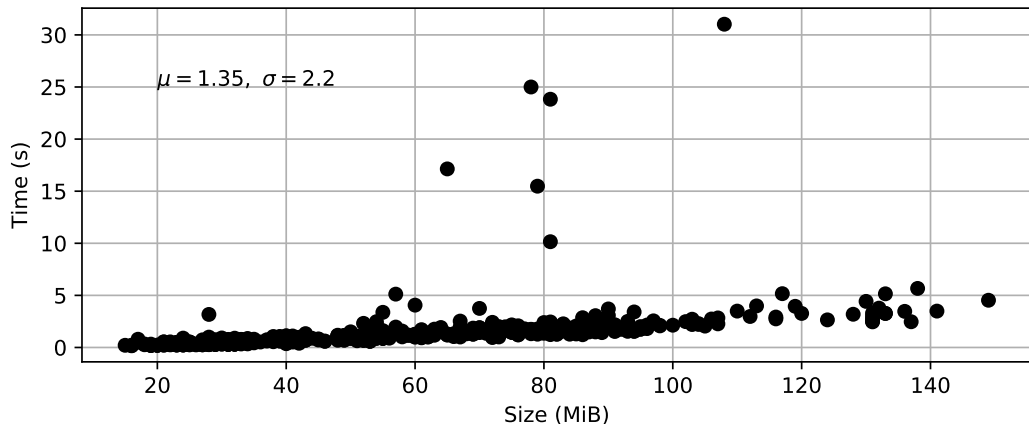


Figure: Time to take the snapshot of an application in relation to its size. Every dot represents one application

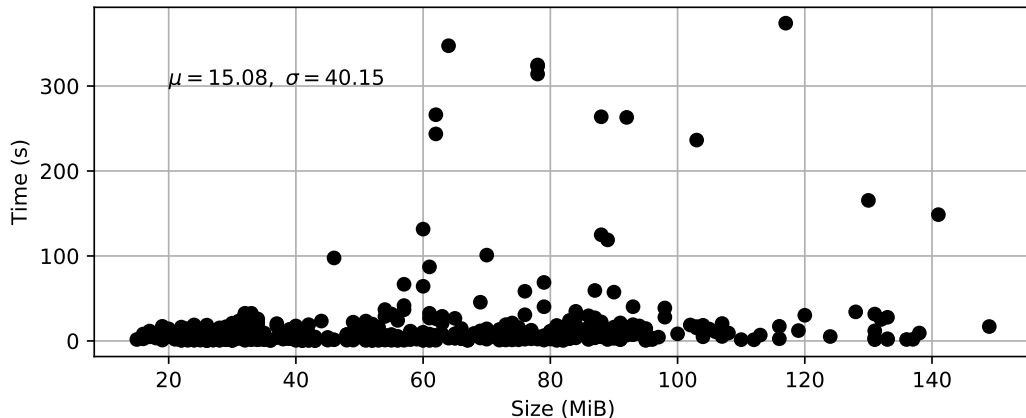


Figure: Time to extract the key material of a snapshot. Every dot represents an application snapshot

Improve performance: Searching the key in memory sequentially is slow!

- ▶ **Idea:** Follow pointers to session key (e.g., pointers of function calls)
- ▶ **Ephemeral data:** When to extract the data?

DroidKex approach:

- ▶ Intercept function calls and extract session keys
- ▶ Find a “path” in the nested data structures from function arguments to the session key

Goal of this talk

- ▶ Present why key extraction is not straight forward

1. Bring TLSKex to Android
2. Improve the performance by directly accessing the session key (not searching it with brute force)
3. Data driven approach to derive data structure layout
4. Evaluate approach

1. Timing

- ▶ Why does it fail?
- ▶ How to detect failures?
- ▶ Fall back solution?

2. Interception

- ▶ How to intercept?
- ▶ Which functions should be intercepted?
- ▶ How to relate network connection to session key?

3. Semantic Gap

- ▶ How does the data structure look like?
- ▶ How to derive the information and bridge the semantic gap?

1. Timing

- ▶ Why does it fail?
- ▶ How to detect failures?
- ▶ Fall back solution?

2. Interception

- ▶ How to intercept?
- ▶ Which functions should be intercepted?
- ▶ How to relate network connection to session key?

3. Semantic Gap

- ▶ How does the data structure look like?
- ▶ How to derive the information and bridge the semantic gap?

Why does control flow based key extraction fail?

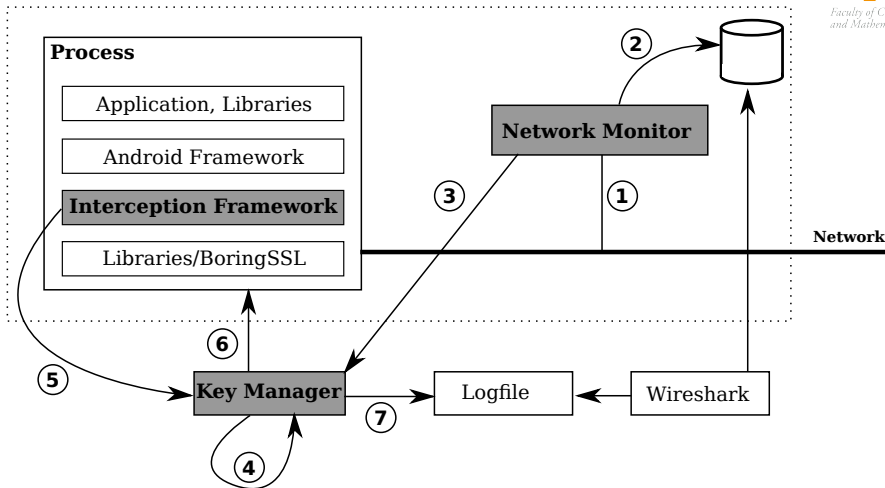
- ▶ Interception does not always work (function unknown, data structure unknown)
- ▶ Data structure layout does not match expected layout
- ▶ No communication after key negotiation

How to detect failures?

- ▶ Detect new TLS connection via network monitoring
- ▶ Wait to receive keys from the interception framework
- ▶ Test if key is valid by decrypting the first TLS record

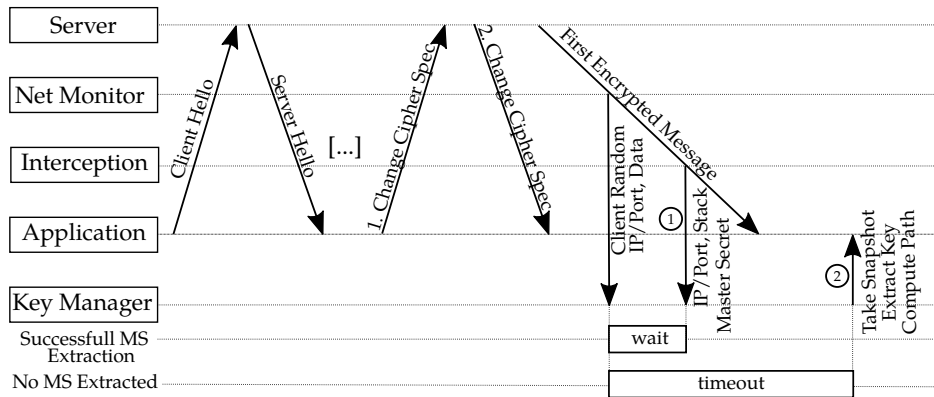
What to do if function tracing does not work?

- ▶ Take a snapshot of the memory
- ▶ Derive data structure layout



Problem: Message 5 can arrive before 3 due to concurrency and communication latency

Timing: Fall back solution - Defining the timeout



Timeout time to wait for key from interception framework.

Trade-off: Too short (high overhead, many snapshots) vs. too late (loss of key)

1. Timing

- ▶ Why does it fail?
- ▶ How to detect failures?
- ▶ Fall back solution?

2. Interception

- ▶ How to intercept?
- ▶ Which functions should be intercepted?
- ▶ How to relate network connection to session key?

3. Semantic Gap

- ▶ How does the data structure look like?
- ▶ How to derive the information and bridge the semantic gap?

- ▶ **Overloading with LD_PRELOAD:**
 - ▶ cross-compiling required
 - ▶ hard to debug
 - ▶ hard to use

- ▶ **Frida:** “Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers”²
 - ▶ easy to use
 - ▶ works with most Android versions
 - ▶ provides useful functionalities
 - ▶ can only be attached to a running process
 - ▶ starting an app with Frida attached did not work for us!

²<https://www.frida.re/>

Requirements

- ▶ Address/Name of function must be known
- ▶ Address must have a parameter that points to the session key
- ▶ Key must be available when function is called

Possible functions

- ▶ `SSL_read` and `SSL_write`
 - ▶ Not always exported or linked static to application
 - ▶ Works only for apps that use BoringSSL/OpenSSL
 - ▶ No correlation to network connection possible
- ▶ `read`, `write`, `send`, `recv`
 - ▶ Address of the function is not required
 - ▶ The pointer of the library function must be on the stack (not exactly known where)
 - ▶ Easy to derive IP/port of network connection from file descriptor

1. Timing

- ▶ Why does it fail?
- ▶ How to detect failures?
- ▶ Fall back solution?

2. Interception

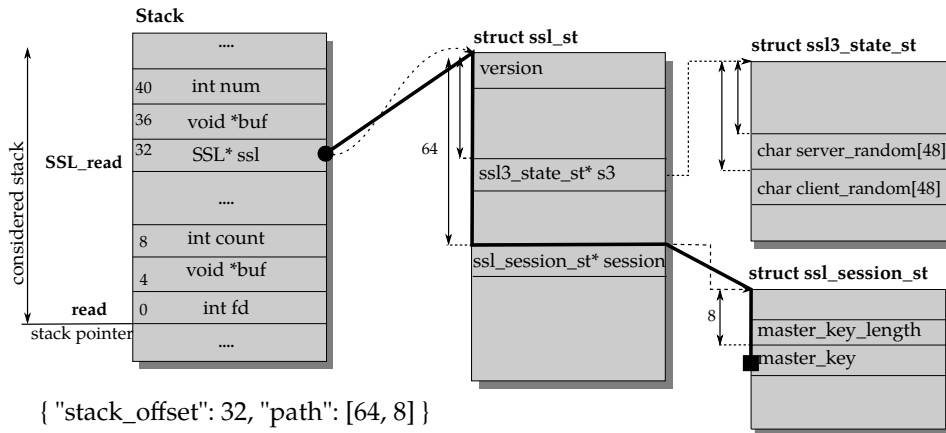
- ▶ How to intercept?
- ▶ Which functions should be intercepted?
- ▶ How to relate network connection to session key?

3. Semantic Gap

- ▶ How does the data structure look like?
- ▶ How to derive the information and bridge the semantic gap?

Semantic Gap: How do the data structures look like?

Example:



Problem: The exact data structure is unknown since it depends on software version and compiler options

Derive from contents in memory!

1. Generate several pairs of <snapshot, stack, session key>
 - ▶ Make application communicate (Touch something, ...)
 - ▶ **Timeout:** no keys have been sent by interception framework
 - create snapshot and save parameters on stack
 - search for keys (TLSKex approach)
2. Start to search paths from the values of the stack to the location of the session key
3. Find the path that occurs in all snapshots
4. **Repeat** until a key was found for each connection, i.e., the interception framework sends always a key and the timeout is never reached

Multiple Paths: An application can use different libraries, TLS versions/configurations

Problem: Approach only works for data structures that have always the same path

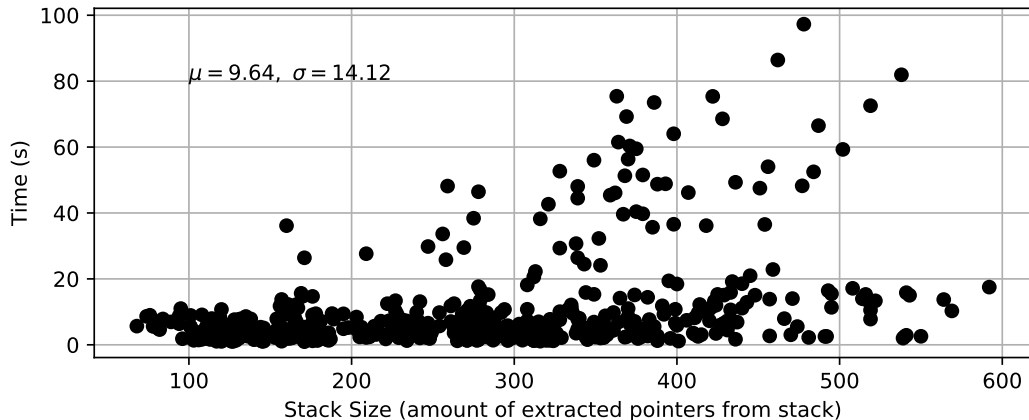


Figure: Time to compute the data path compared to the size of distinct dereferenceable pointer on the stack collected from I/O calls

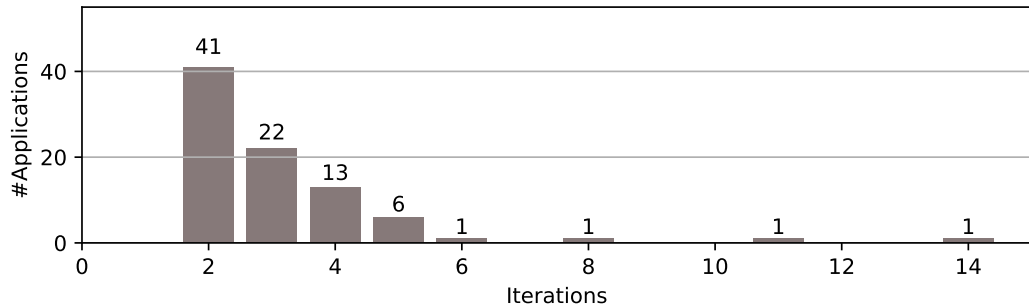


Figure: Number of iterations required to compute paths

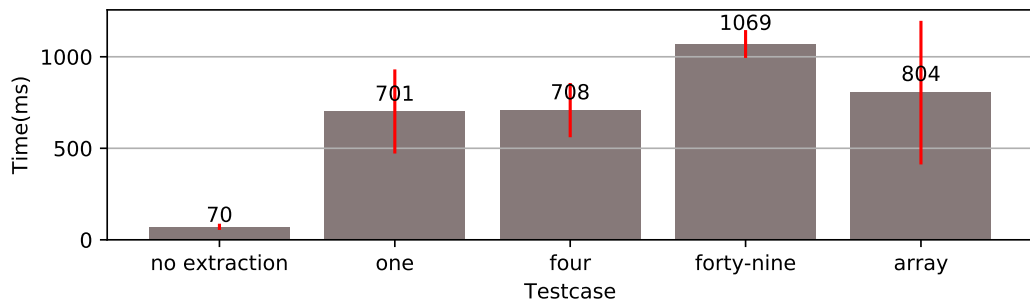
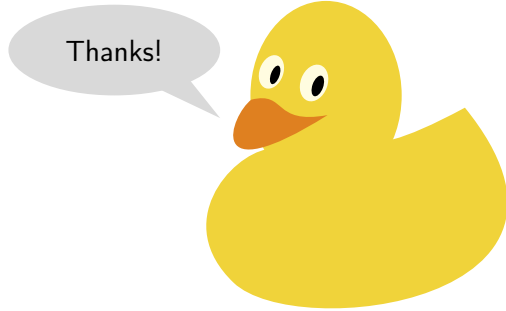


Figure: Time for a single TLS https GET request with a different number of extraction paths

Time to take snapshot (avg): 1.35 s

Time to extract key (avg): 15.08 s

- ▶ **Lesson learnt:** Fast key extraction is challenging!
- ▶ **Semantic Gap:** The data driven approach of DroidKex can be used to bridge the semantic gap in other use cases
- ▶ **Future Work:**
 - ▶ Improve the performance and minimize the interception overhead
 - ▶ Derive semantic knowledge from complex data structures (array, list, hashmap, ...)
 - ▶ Publish the code!



Thanks!