# 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

DINGfest
DetektIon, VisualisieruNG, ForEnsische Aufbereitung von SicherheiTsvorfällen

Bundesministerium für Bildung und Forschung

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

**Goal:** Decrypt TLS encrypted communication channels for Android applications

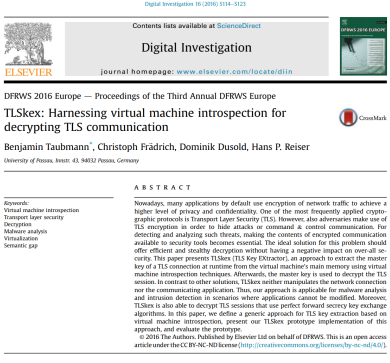
**TLS - Perfect Forward Secrecy (PFS)**

- 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

# Prior work: TLSkex[1]



Digital Investigation 16 (2016) S114–S123

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2016 Europe — Proceedings of the Third Annual DFRWS Europe

TLSkex: Harnessing virtual machine introspection for decrypting TLS communication

Benjamin Taubmann*, Christoph Frädrich, Dominik Dusold, Hans P. Reiser

*University of Passau, Innstr. 43, 94032 Passau, Germany*

ABSTRACT

Nowadays, many applications by default use encryption of network traffic to achieve a higher level of privacy and confidentiality. One of the most frequently applied cryptographic protocols is Transport Layer Security (TLS). However, also adversaries make use of TLS encryption in order to hide attacks or command & control communication. For detecting and analyzing such threats, making the contents of encrypted communication available to security tools becomes essential. The ideal solution for this problem should offer efficient and stealthy decryption without having a negative impact on over-all security. This paper presents TLSkex (TLS Key EXtractor), an approach to extract the master key of a TLS connection at runtime from the virtual machine's main memory using virtual machine introspection techniques. Afterwards, the master key is used to decrypt the TLS session. In contrast to other solutions, TLSkex neither manipulates the network connection nor the communicating application. Thus, our approach is applicable for malware analysis and intrusion detection in scenarios where applications cannot be modified. Moreover, TLSkex is also able to decrypt TLS sessions that use perfect forward secrecy key exchange algorithms. In this paper, we define a generic approach for TLS key extraction based on virtual machine introspection, present our TLSkex prototype implementation of this approach, and evaluate the prototype.



*Keywords:*
Virtual machine introspection
Transport layer security
Decryption
Malware analysis
Virtualization
Semantic gap

## 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

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

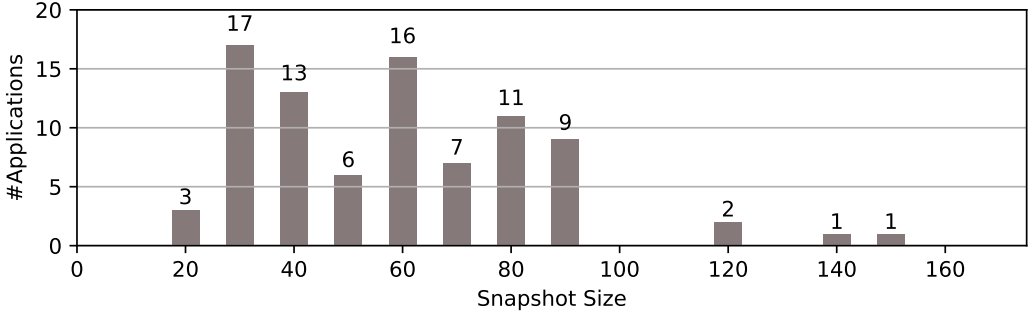



Figure: **Histogram:** Number of applications with the same snapshot memory size (Size in MiB)

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

# Time to take snapshot

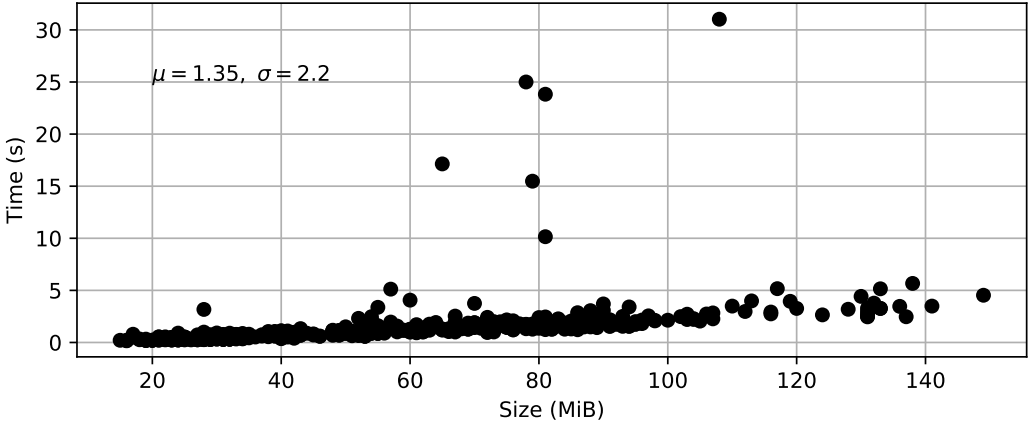



Figure: Time to take the snapshot of an application in relation to its size. Every dot represents one application

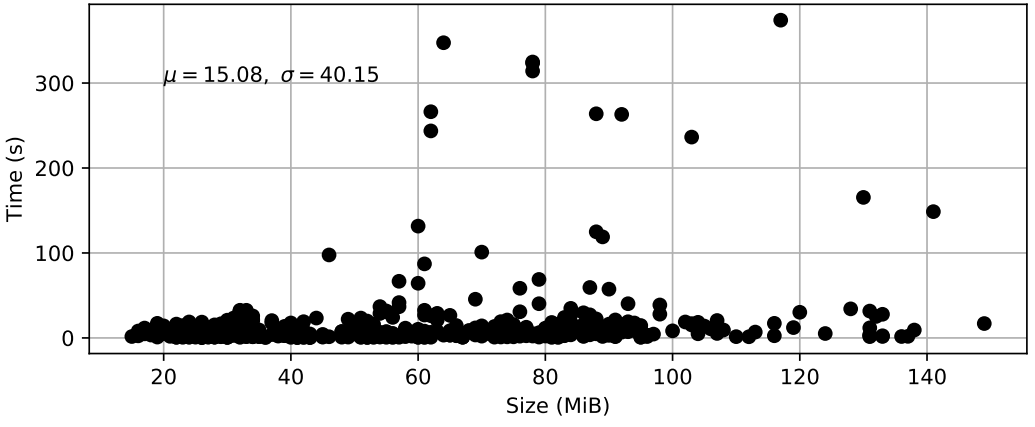



Figure: Time to extract the key material of a snapshot. Every dot represents an application snapshot

**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

# DroidKex: Contributions

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

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?

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

**Why does control flow based key extraction fail?**

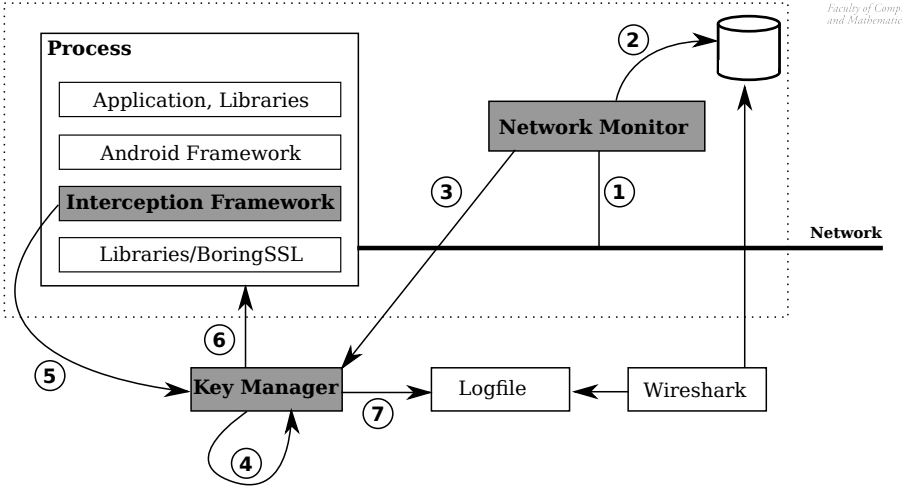
- Interception does not always work (function unknown, data structure unknown)
- Data structure layout does not match expected layout
- No communication after key negotiation

**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

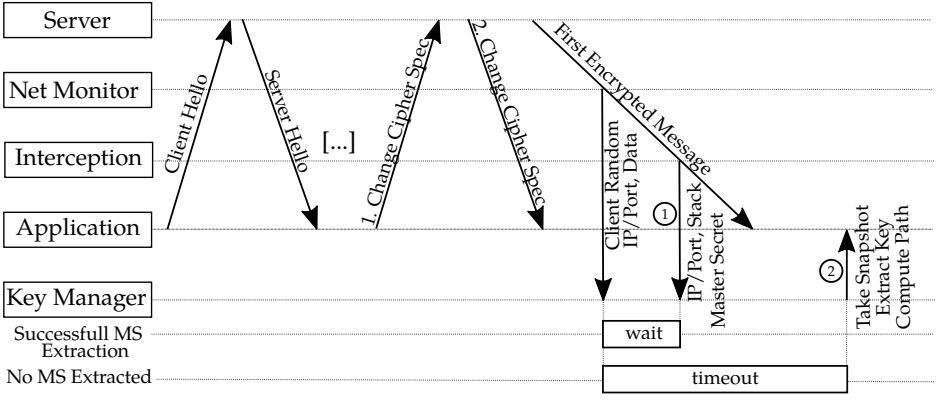
**What to do if function tracing does not work?**

- 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/

# Interception: Which functions can be intercepted?

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

**Requirements**

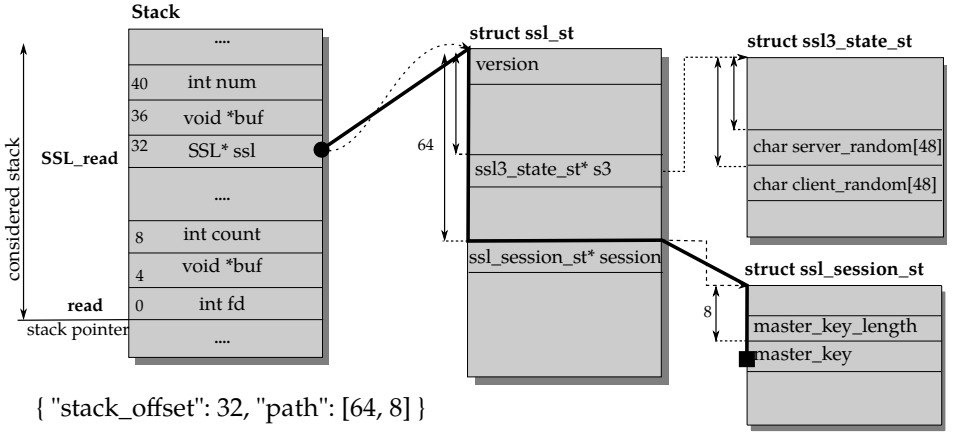
- Address/Name of function must be known
- Address must have a parameter that points to the session key
- Key must be available when function is called

**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?

**Example:**




{ "stack_offset": 32, "path": [64, 8] }

**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

# Evaluation: Time to compute single path

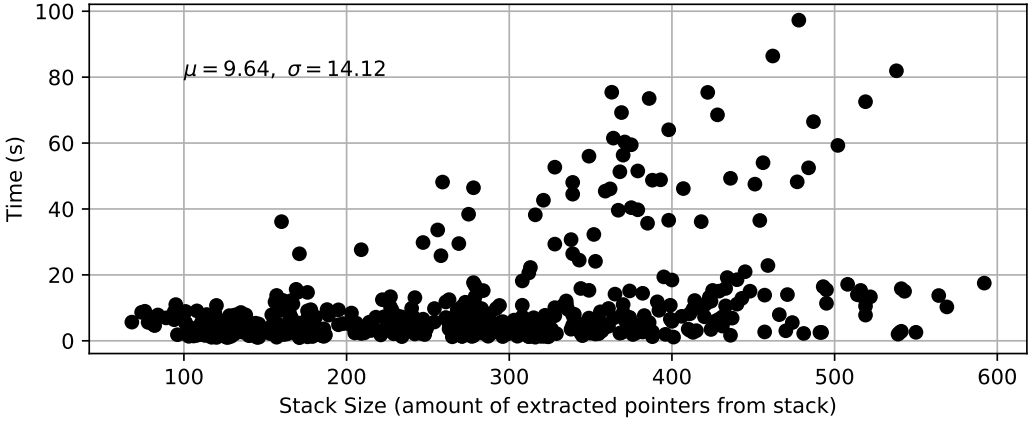



Figure: Time to compute the data path compared to the size of distinct dereferenceable pointer on the stack collected from I/O calls

# Evaluation: Path computation – Required Iterations

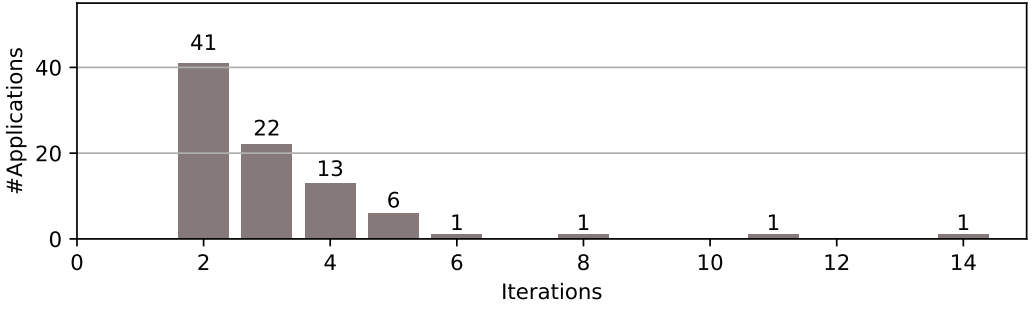



Figure: Number of iterations required to compute paths

# Evaluation: Connection overhead

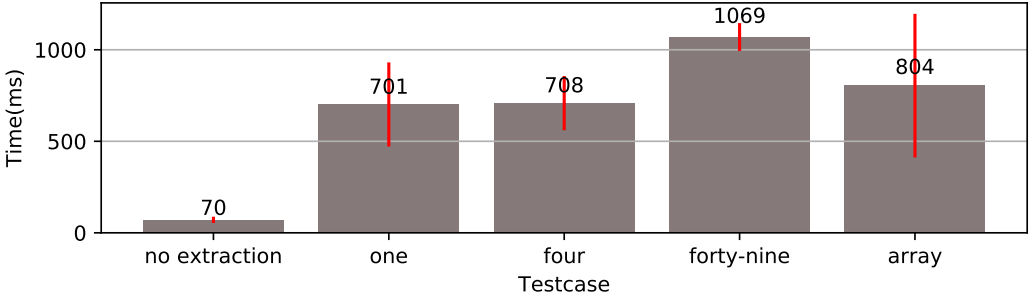


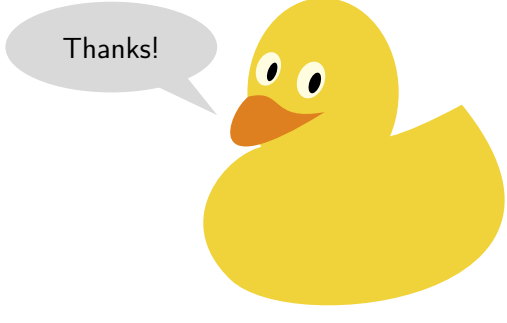

Figure: Time for a single TLS https GET request with a different number of extraction paths

**Time to take snapshot (avg):** 1.35 s
**Time to extract key (avg):** 15.08 s

# Conclusion

UNIVERSITY OF PASSAU
Faculty of Computer Science and Mathematics

- **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!