Dalvik Memory Analysis and a Call to ARMs

Joe T. Sylve, M.S.

Managing Partner

504ENSICS Labs

Android Memory Analysis

504ENSICS LABS

- Acquisition – LiME
  - LKM that can dump an image of physical memory over TCP or to disk
  - Developed by me starting in 2011
  - http://code.google.com/p/lime-forensics
- Kernel Level Analysis
  - Volatility support added in 2012
  - Most of the hard work was done by Andrew Case
  - I contributed and rewrote the ARM address space
- You've heard me talk about this stuff before

Dalvik Analysis

- Dalvik is the Android Runtime for Userland Apps
- Basically Google's Custom JVM
  - That's Why Oracle Sued Them
- Android Applications Each Have Their Own Instance of a DVM
- Kernel Level Analysis
  - Running Processes
  - Network Connections
  - Loaded Kernel Modules
  - Dump Process Heap
    - String and Grep
  - Not Optimal for Analyzing Userland Malware

Dalvik Analysis

- 504ENSICS Labs Awarded Research Grant in Late 2012
  - DARPA Cyber Fast Track Program -- RIP :-(
  - Application-Level Memory Forensics for Dalvik
  - Added Volatility Support for Parsing Objects & Instances from DVM
  - Created a GUI Application (Dalvik Inspector)
    - Easy Analysis of Android Apps
    - Search and Filter Class Names and Fields
    - Automatically Generate Simple Volatility Plugins

How it works from 10,000ft

- Locate Dalvik in Memory
  - libdvm.so
  - Linux C Shared Library
  - Loaded into Every Dalvik Process
  - By performing analysis of its data structures in memory, we can recreate the entire state of an application at the time of the memory capture
  - Can access this with existing Volatility functionality

How it works from 10,000ft

- Finding Loaded Classes
  - *DvmGlobals gDvm*
    - Global variable inside of libdvm.so
    - Declared in *vm/init.c* of source
    - Hold the members that are needed to located classes in memory
      - Enumerate the *loadedClasses* member
        - Type is *HashTable*
        - Elements are of type *ClassObject*
        - Static and Instance Fields & Methods

How it works from 10,000ft

- Parsing Fields
  - *ClassObject*
    - int *sfieldCount*
    - *StaticField sfields*
    - int *ifieldCount*
    - *InstField* ifields

How it works from 10,000ft

```
struct StaticField {
    Field     field;    /* MUST be first item */
    JValue    value; /* initially set from DEX for primitives */
};

struct Field {
    ClassObject* clazz;
    const char*  name;
    const char*  signature;
    u4           accessFlags;
};

typedef union JValue {
    u1      z;
    s1      b;
    u2      c;
    s2      s;
    s4      i;
    s8      j;
    float   f;
    double  d;
    void*   l;
} JValue;

struct InstField {
    Field  field;
    int    byteOffset;
};
```

Signature	Type
L	Object
Z	Boolean
C	Char
F	Float
D	Double
B	Byte
S	Short
I	Integer
J	Long
L	Void (Pointer)
[	Array

Offset of *JValue* for specific instance
Relative to the *ClassObject* struct

www.504ensics.com

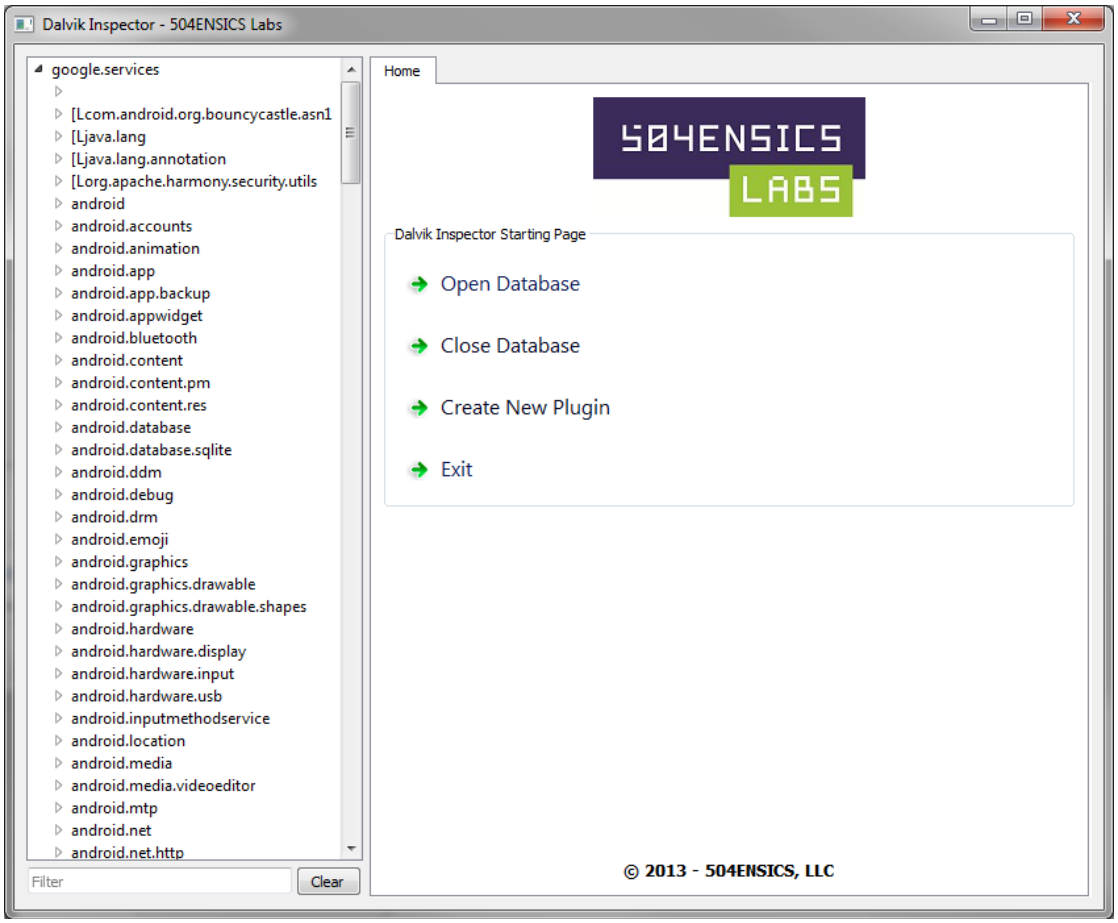
How it works from 10,000ft

- Strings, Arrays, Lists, Objects, etc
  - Slightly more complicated to parse
  - Not terribly difficult
  - I'm probably already running low on time though…

The Process

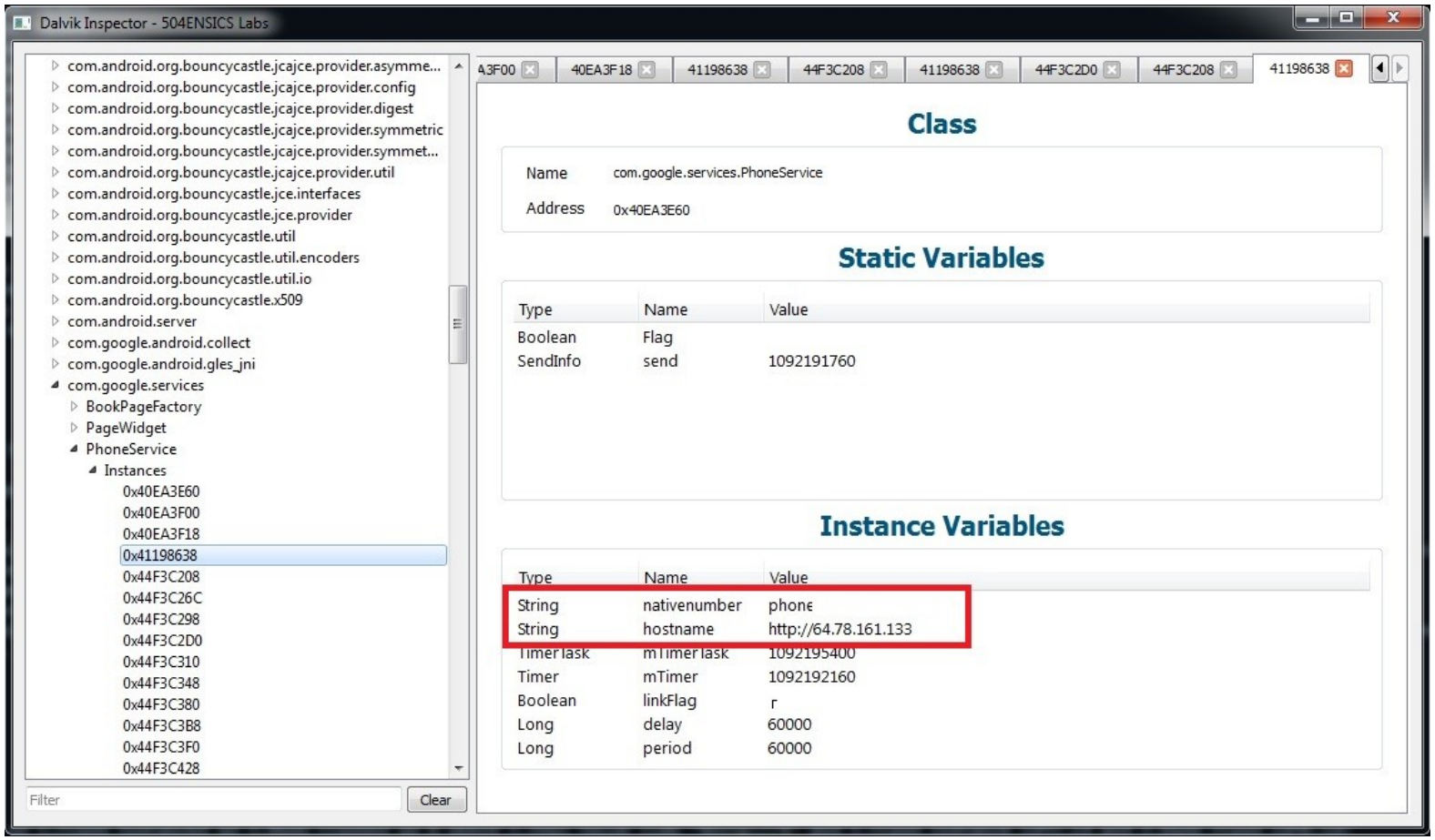
- Infect Device or Emulator
- Dump Memory with LiME
- Run Volatility Plugin to Generate SQLiteDB
- Analyze DB in Dalvik Inspector

Dalvik Inspector



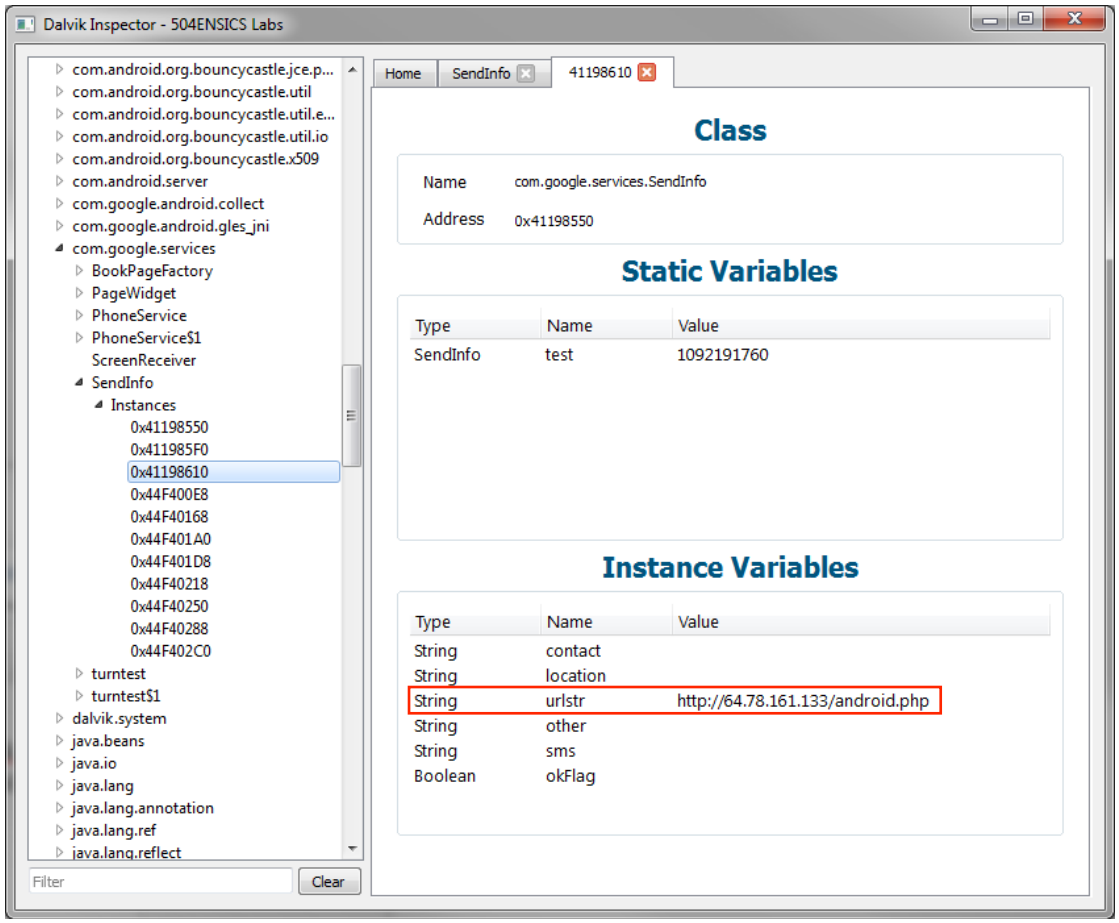
www.504ensics.com

Dalvik Inspector



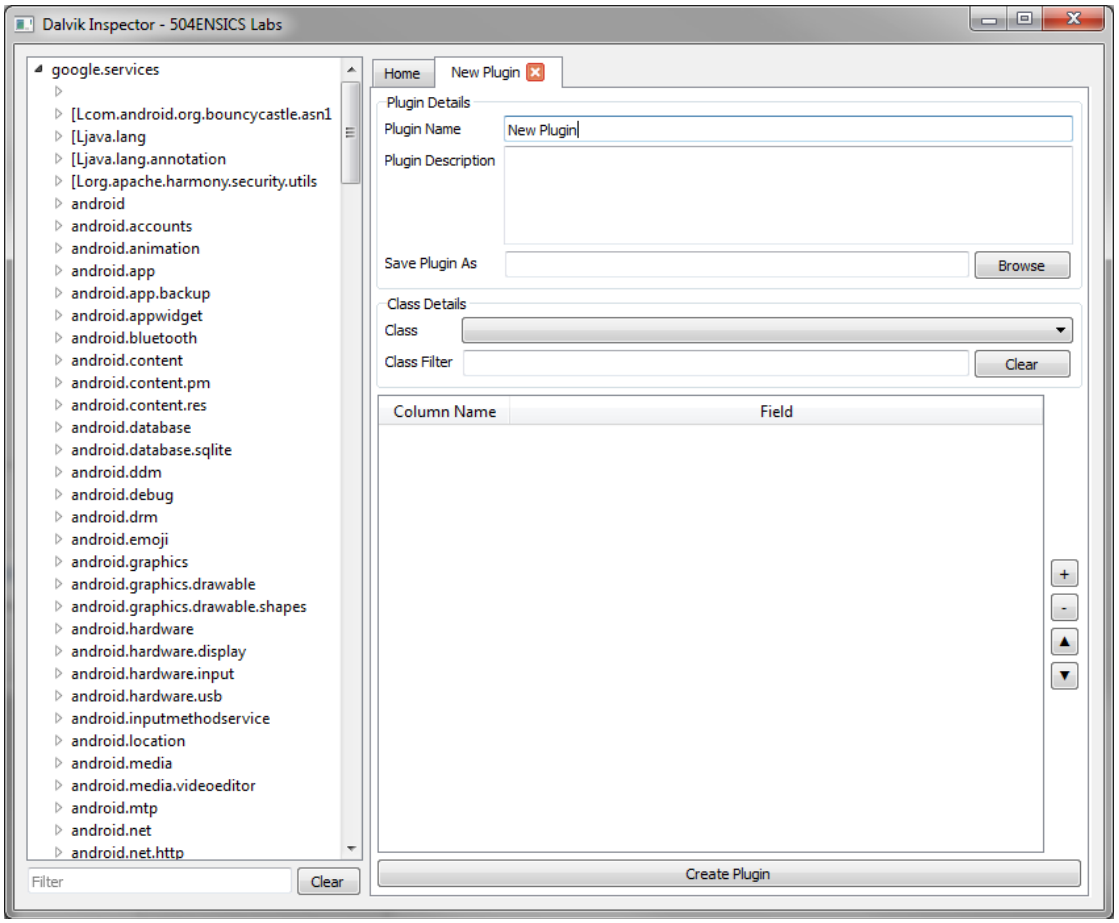
www.504ensics.com

C&C Address



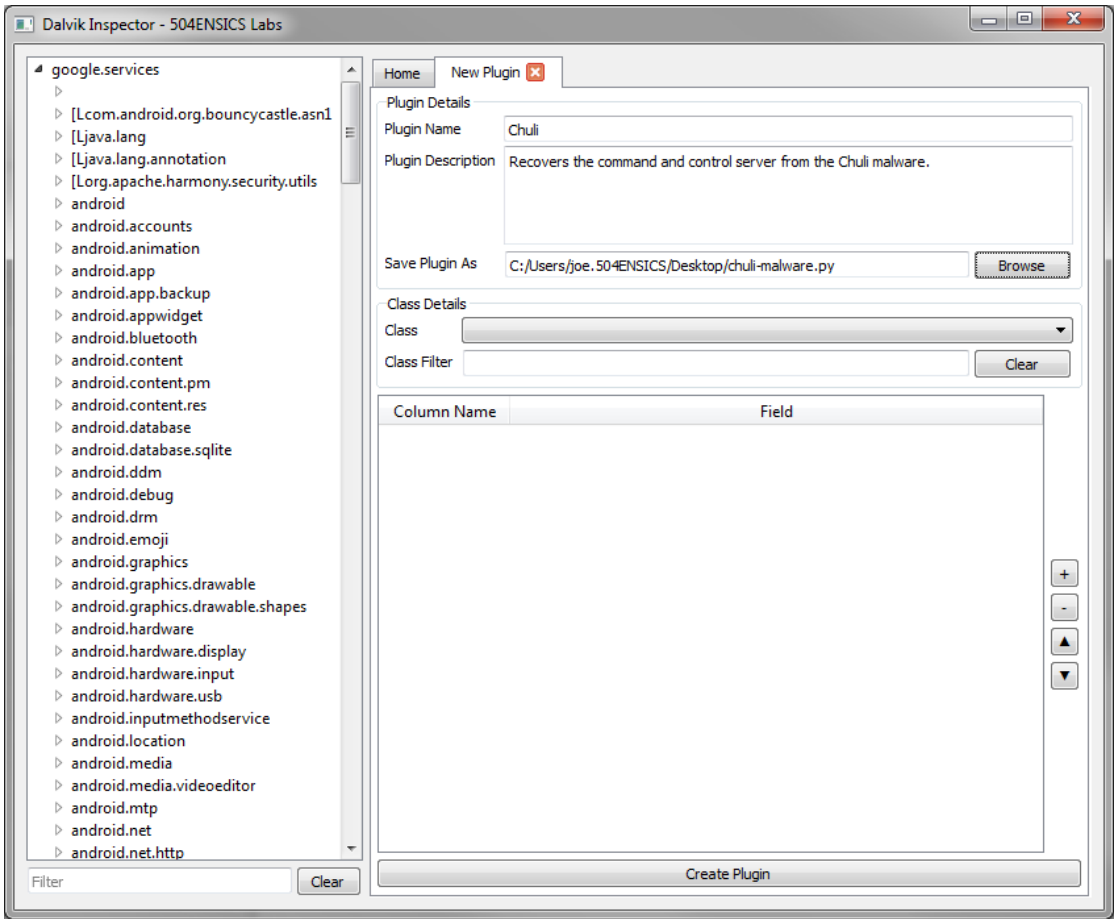
www.504ensics.com

Plugin Creation



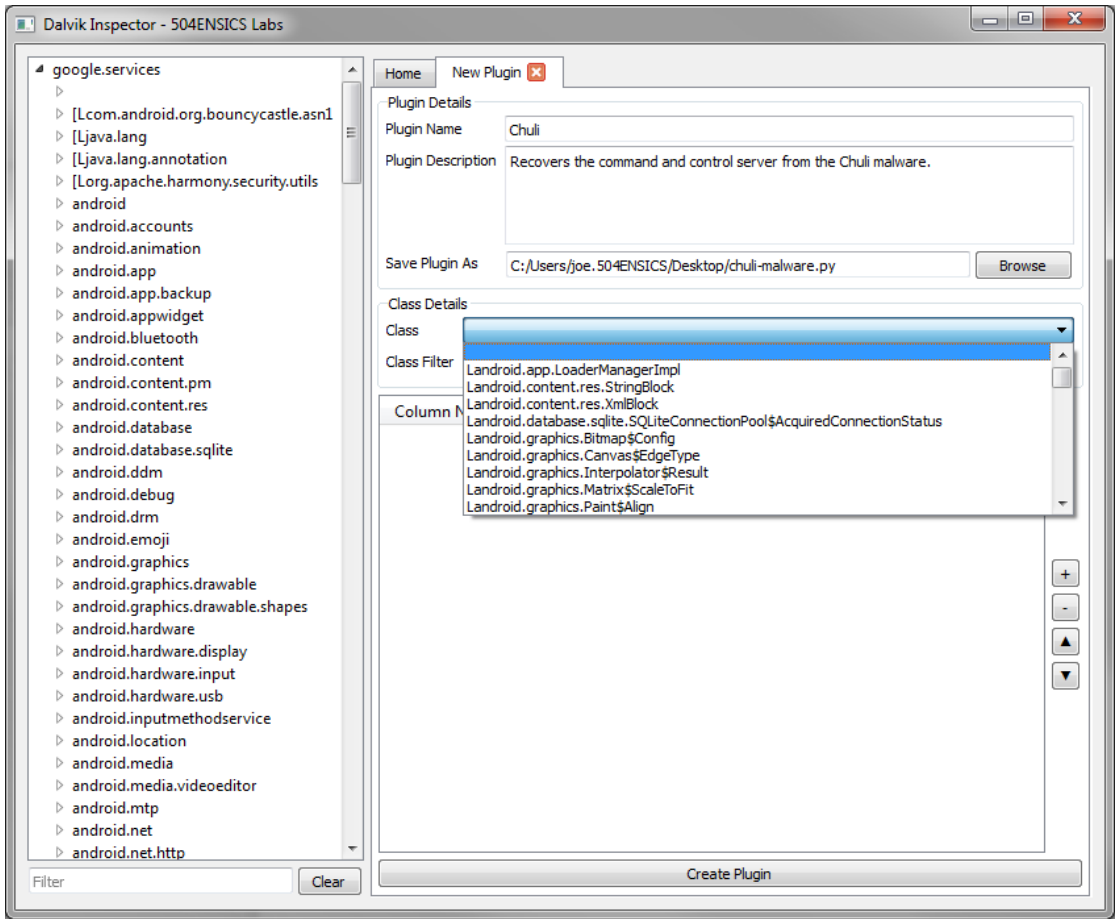
www.504ensics.com

Plugin Creation



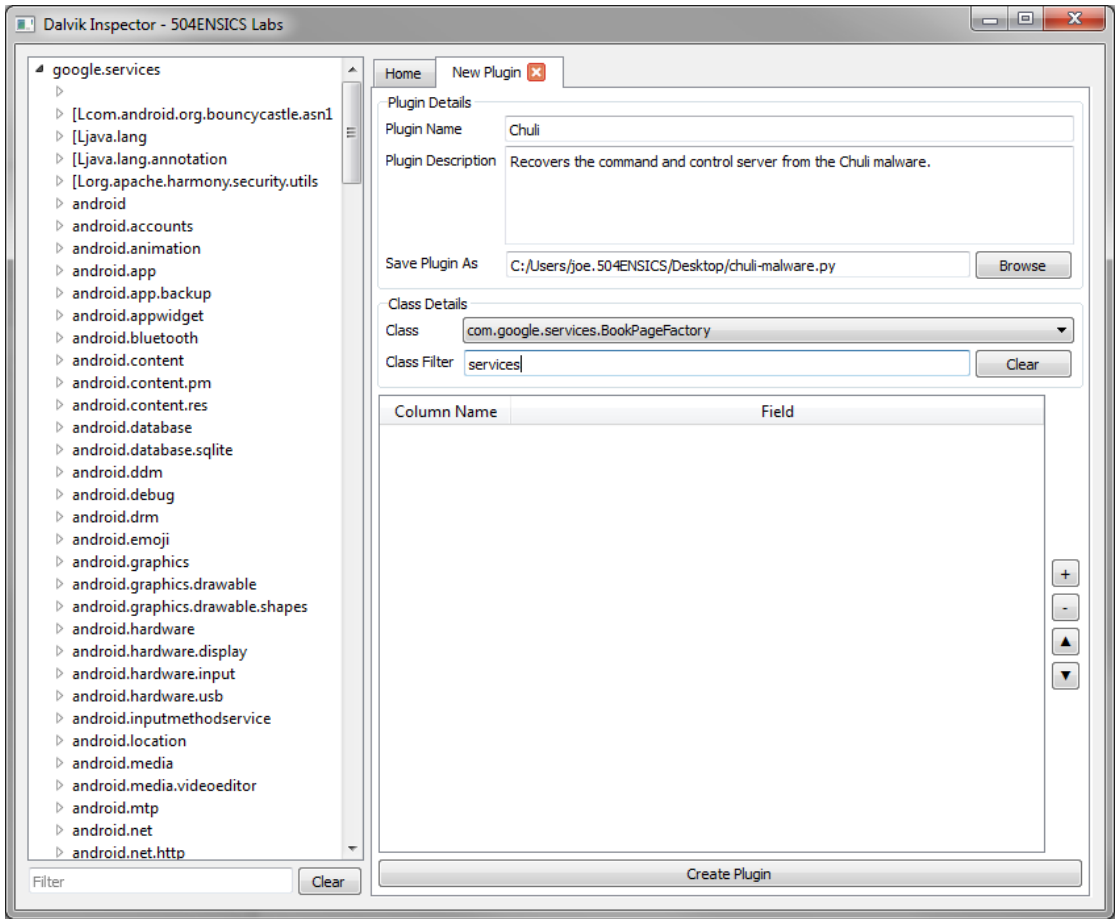
www.504ensics.com

Plugin Creation



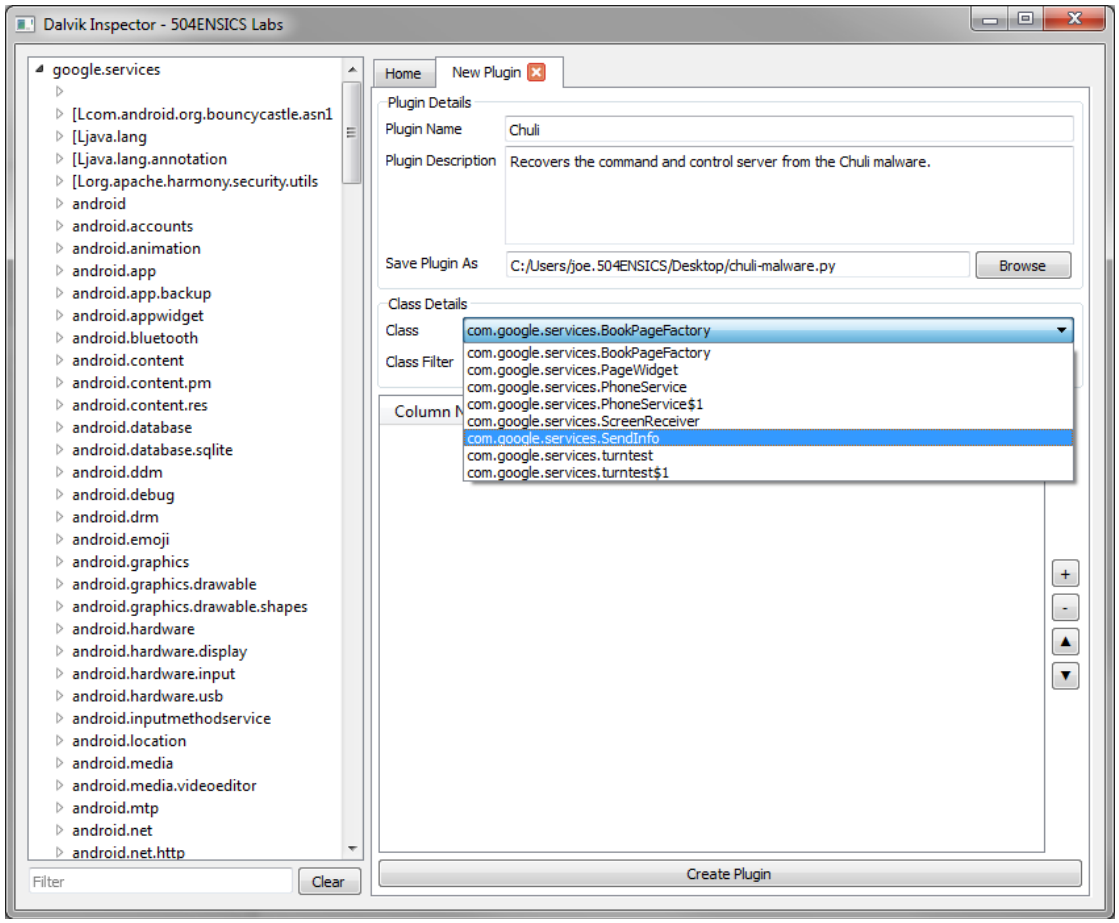
www.504ensics.com

Plugin Creation



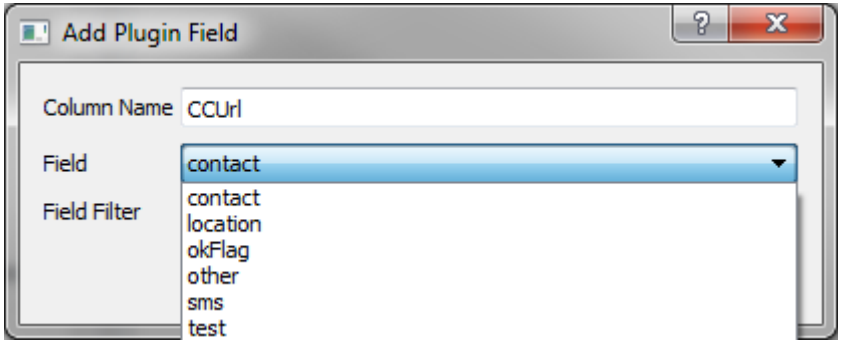
www.504ensics.com

Plugin Creation

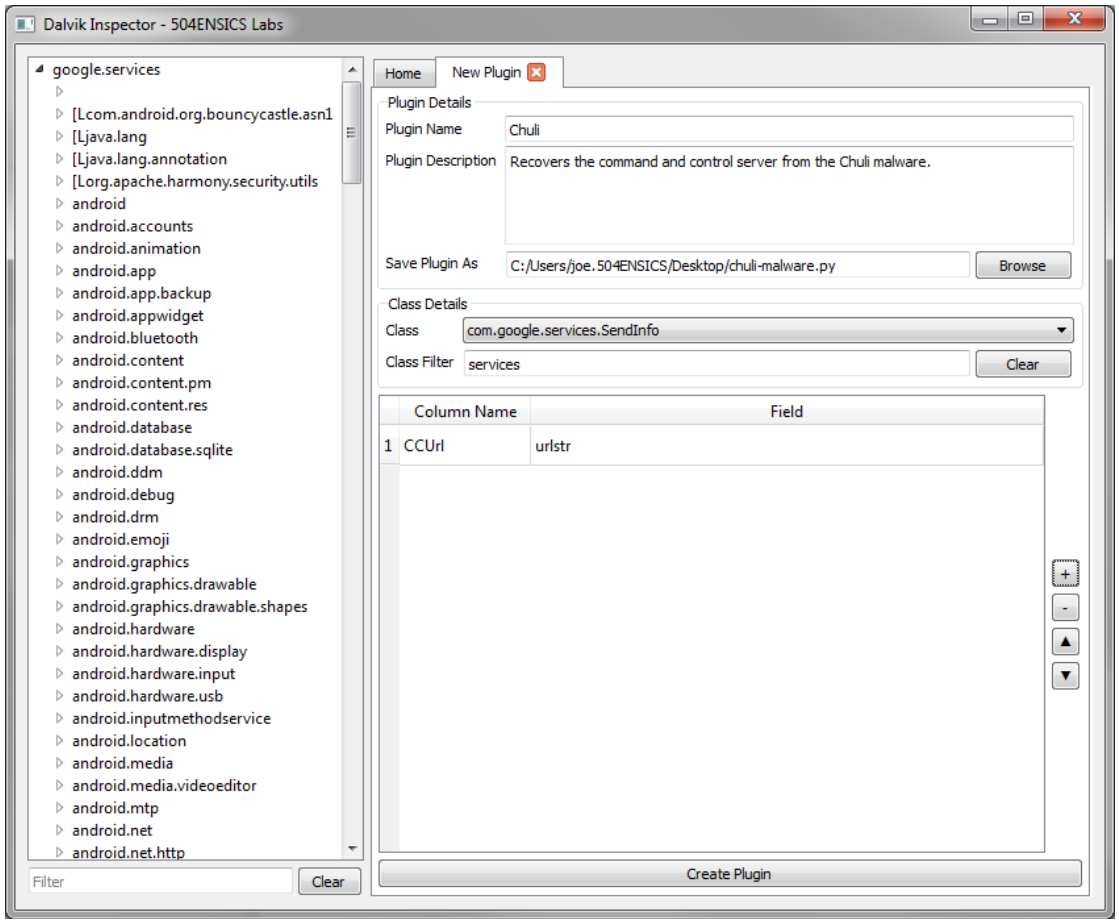


www.504ensics.com

Plugin Creation

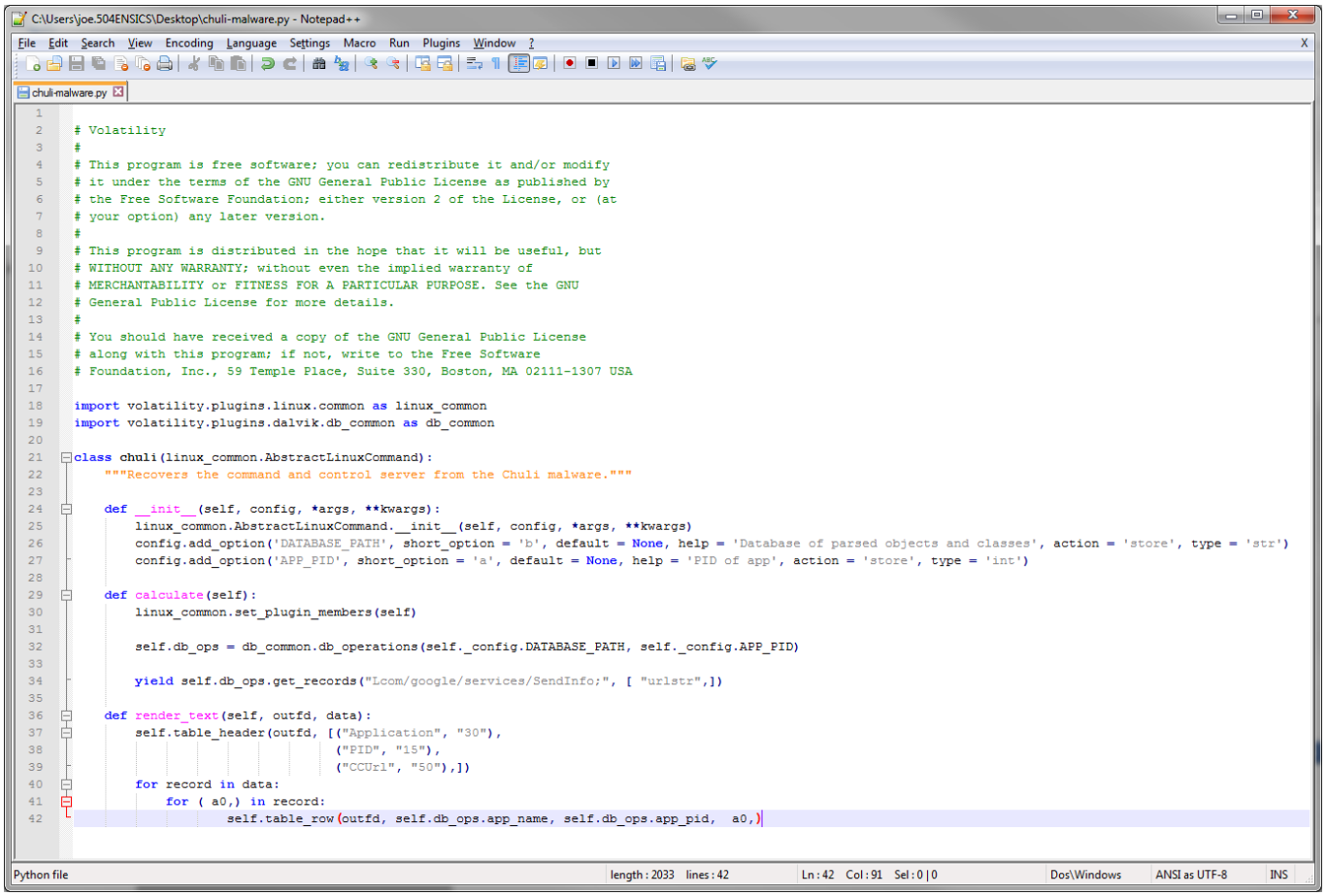


Plugin Creation



www.504ensics.com

Plugin Creation



www.504ensics.com

Plugin Creation

```
root@android64:~/voldalvik/voldalvik# python vol.py --profile=Linuxemulatorx86 -f ../in.lime chuli -a 997 -b /root/chuli-sample.db
Volatile Systems Volatility Framework 2.3_alpha
Application                    PID             CCUrl
------------------------------ --------------- ------------------------------------------------------------
google.services                            997 0
google.services                            997 http://64.78.161.133/android.php
```

Open Research Problems

- Portable LiME LKM
  - Requires Kernel headers and recompiling for each kernel
- Profile Generation
  - Kernel
    - Kernel Headers
    - Exact Config
    - System.map
  - libdvm.so
    - Work in progress
      - Radare Library
- Solution?
  - Combination of static and dynamic analysis
  - Locate symbols
  - Infer config for “key” structures

Questions?

- Dalvik Inspector will be released soon™
- Email for access to closed Beta
  - joe@504Labs.com