



Dalvik Memory Analysis and a Call to ARMs

Joe T. Sylve, M.S.
Managing Partner
504ENSICS Labs

Android Memory Analysis

- 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

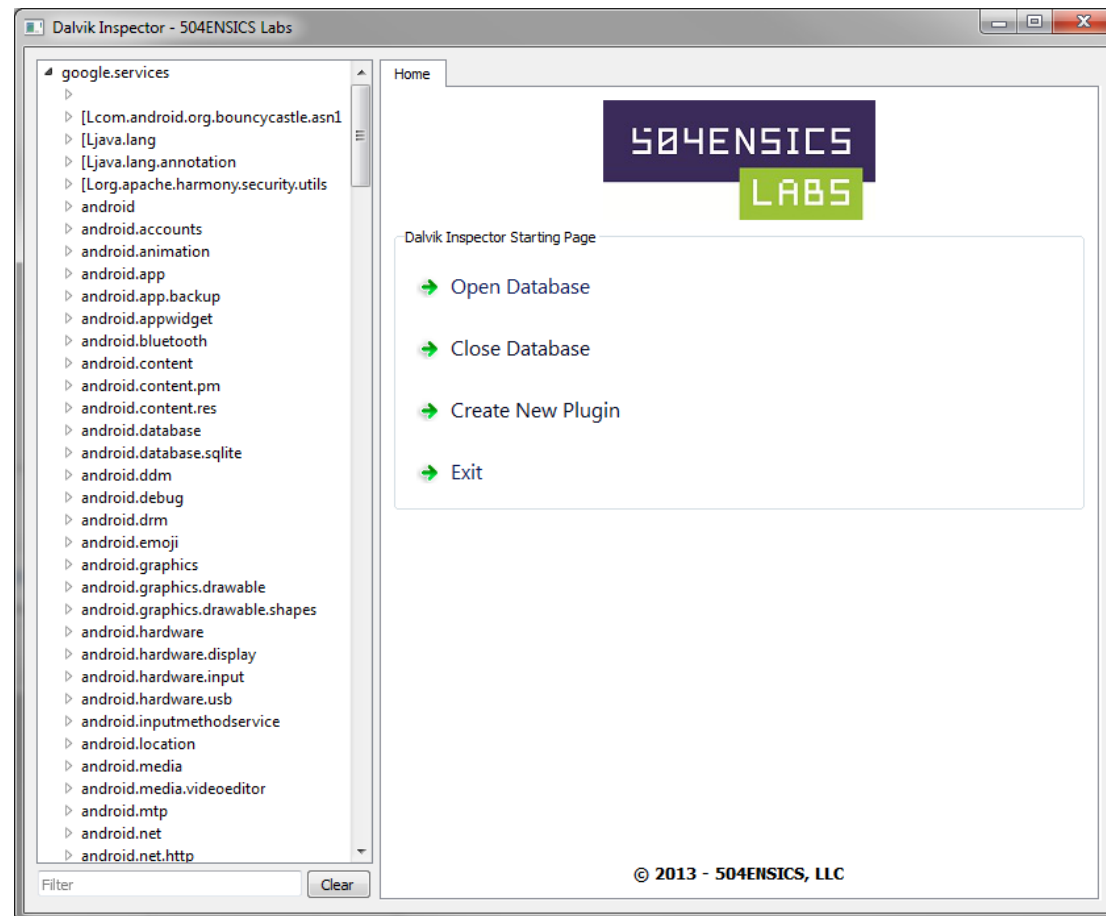
How it works from 10,000ft

- 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



Dalvik Inspector

The screenshot shows the Dalvik Inspector interface. On the left, a tree view displays the class hierarchy, with `com.google.services.PhoneService` selected. The main area shows the details for this class, including its name and address. Below this, the static variables and instance variables are listed in tables. The instance variables table has a red box highlighting the `nativenumber` and `hostname` entries.

Class

Name: `com.google.services.PhoneService`
Address: `0x40EA3E60`

Static Variables

Type	Name	Value
Boolean	Flag	
SendInfo	send	1092191760

Instance Variables

Type	Name	Value
String	nativenumber	phone
String	hostname	http://64.78.161.133
TimerTask	mTimerTask	1092195400
Timer	mTimer	1092192160
Boolean	linkFlag	r
Long	delay	60000
Long	period	60000

C&C Address

The screenshot shows the Dalvik Inspector interface. On the left, a class hierarchy is displayed with 'com.google.services.SendInfo' selected. The main panel shows the details for this class, including its name, address, and static variables. Below, the instance variables are listed, with the 'urlstr' variable highlighted in red, showing the value 'http://64.78.161.133/android.php'.

Class

Name	com.google.services.SendInfo
Address	0x41198550

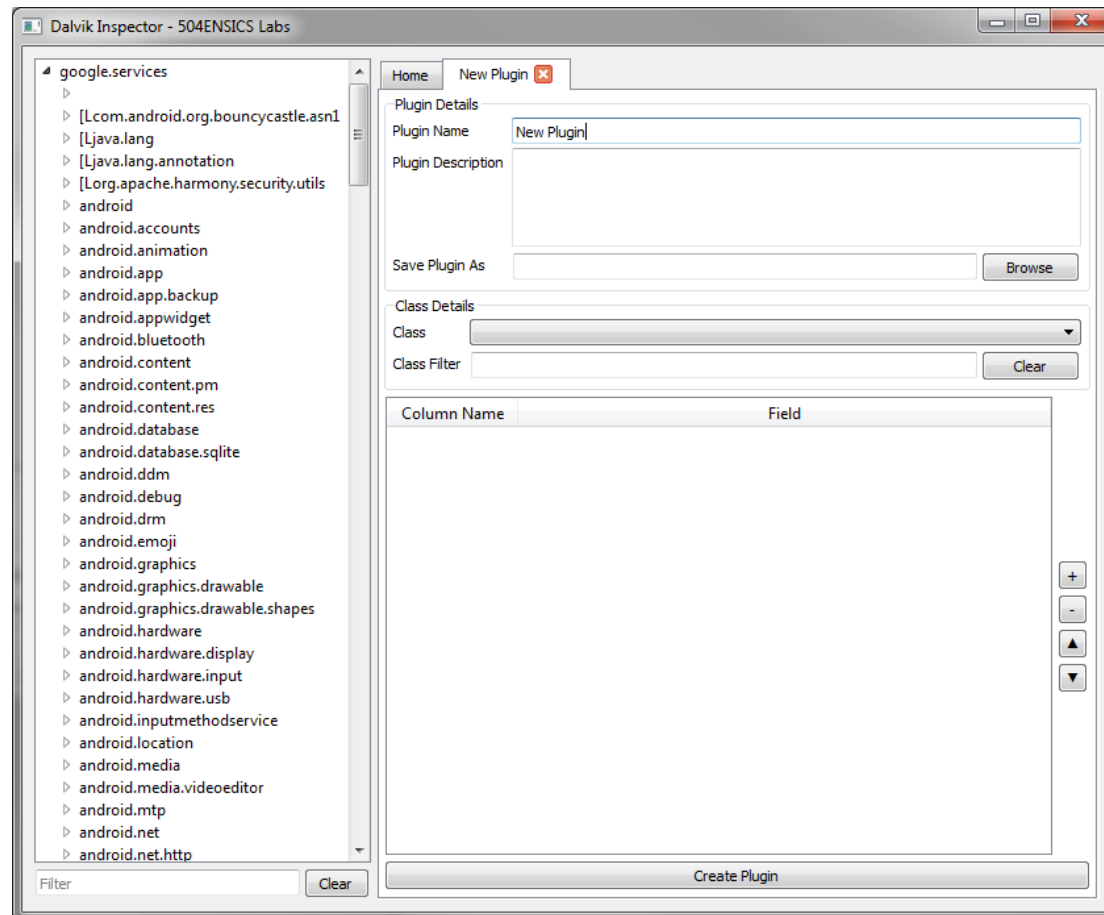
Static Variables

Type	Name	Value
SendInfo	test	1092191760

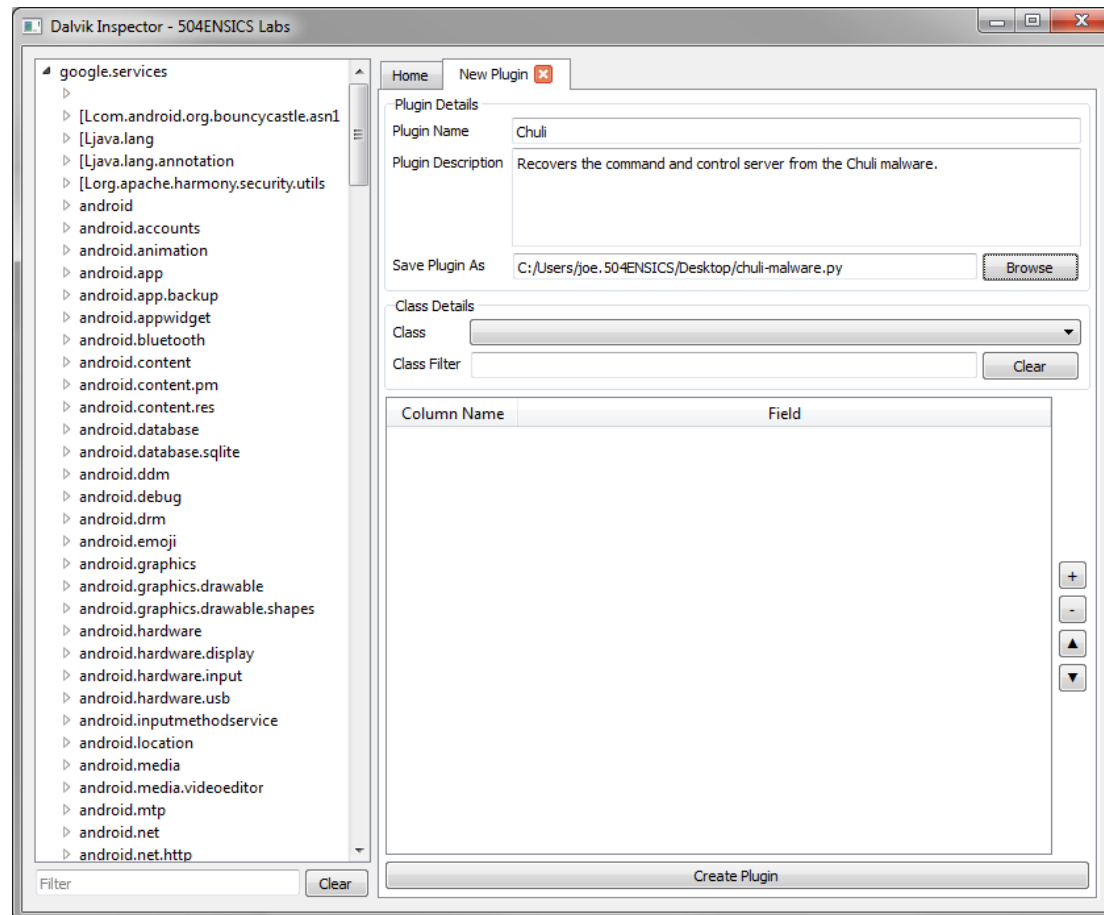
Instance Variables

Type	Name	Value
String	contact	
String	location	
String	urlstr	http://64.78.161.133/android.php
String	other	
String	sms	
Boolean	okFlag	

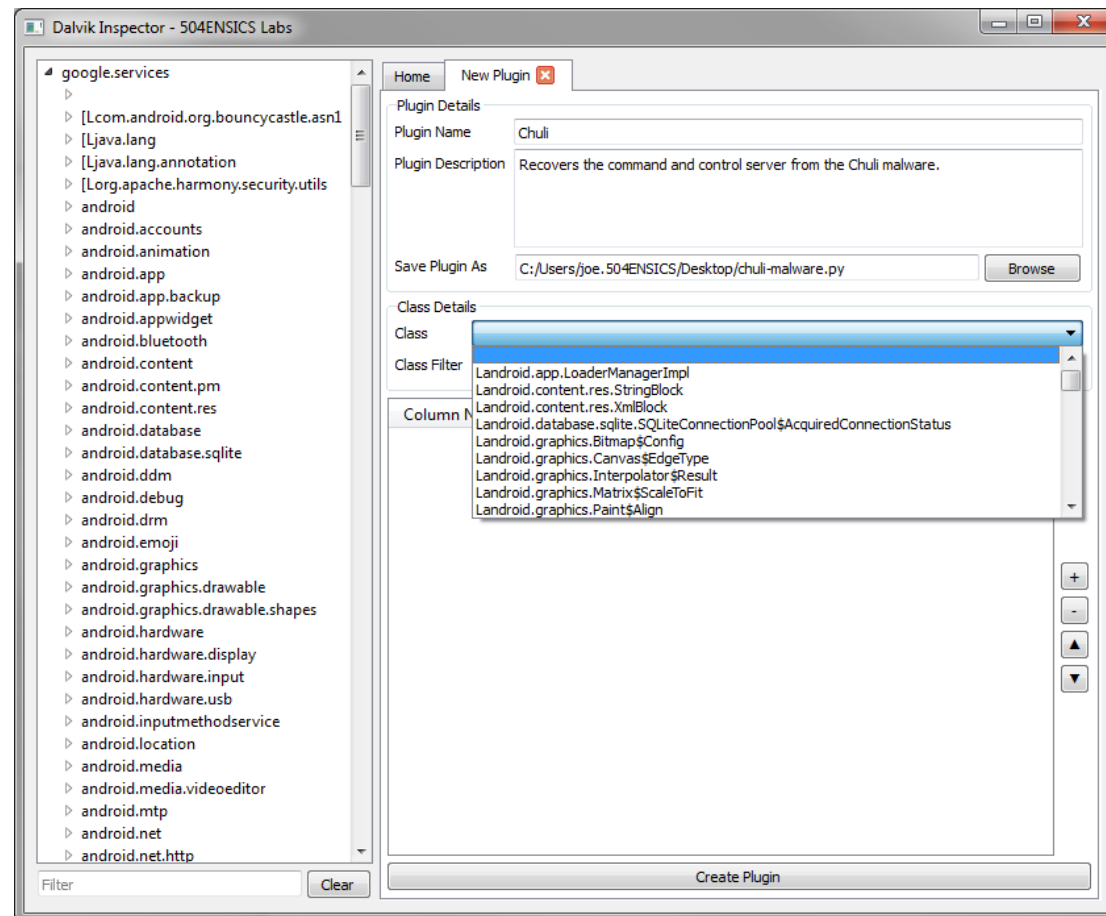
Plugin Creation



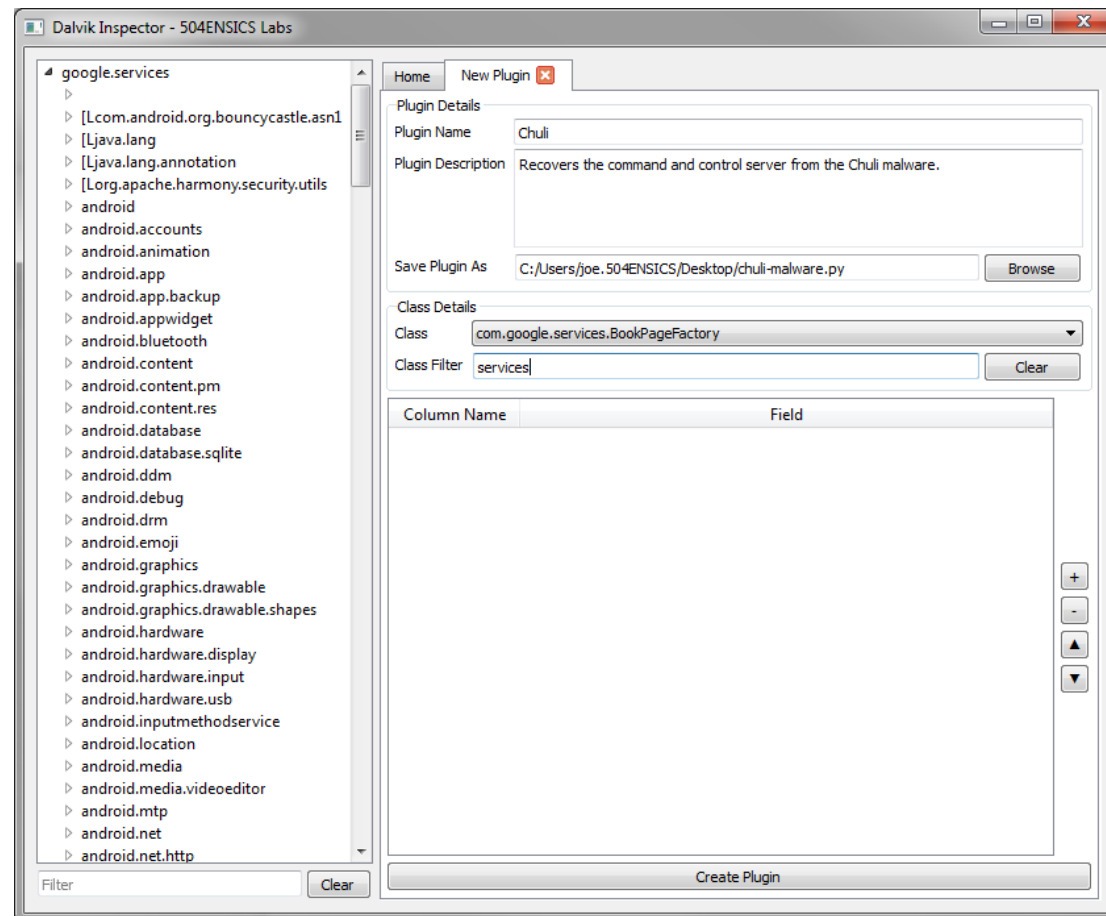
Plugin Creation



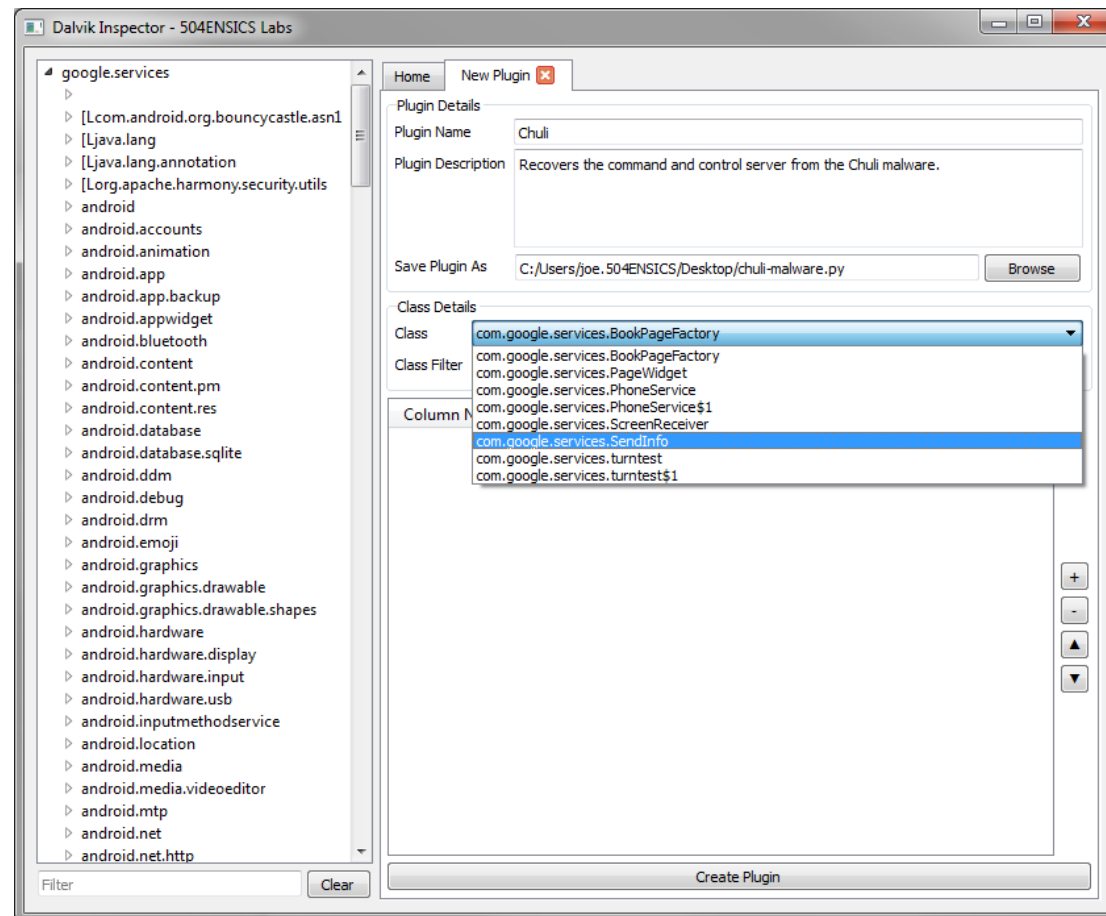
Plugin Creation



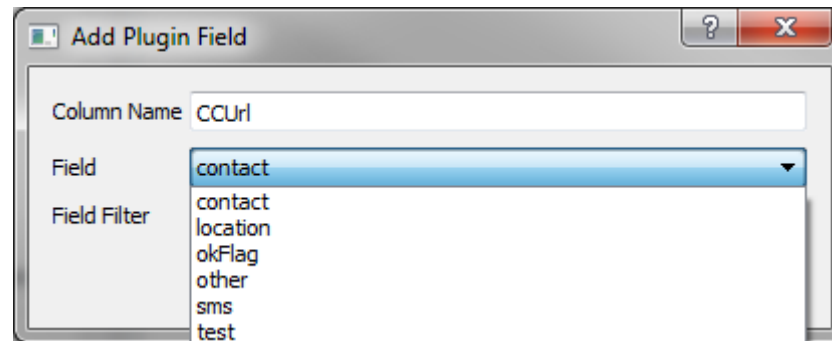
Plugin Creation



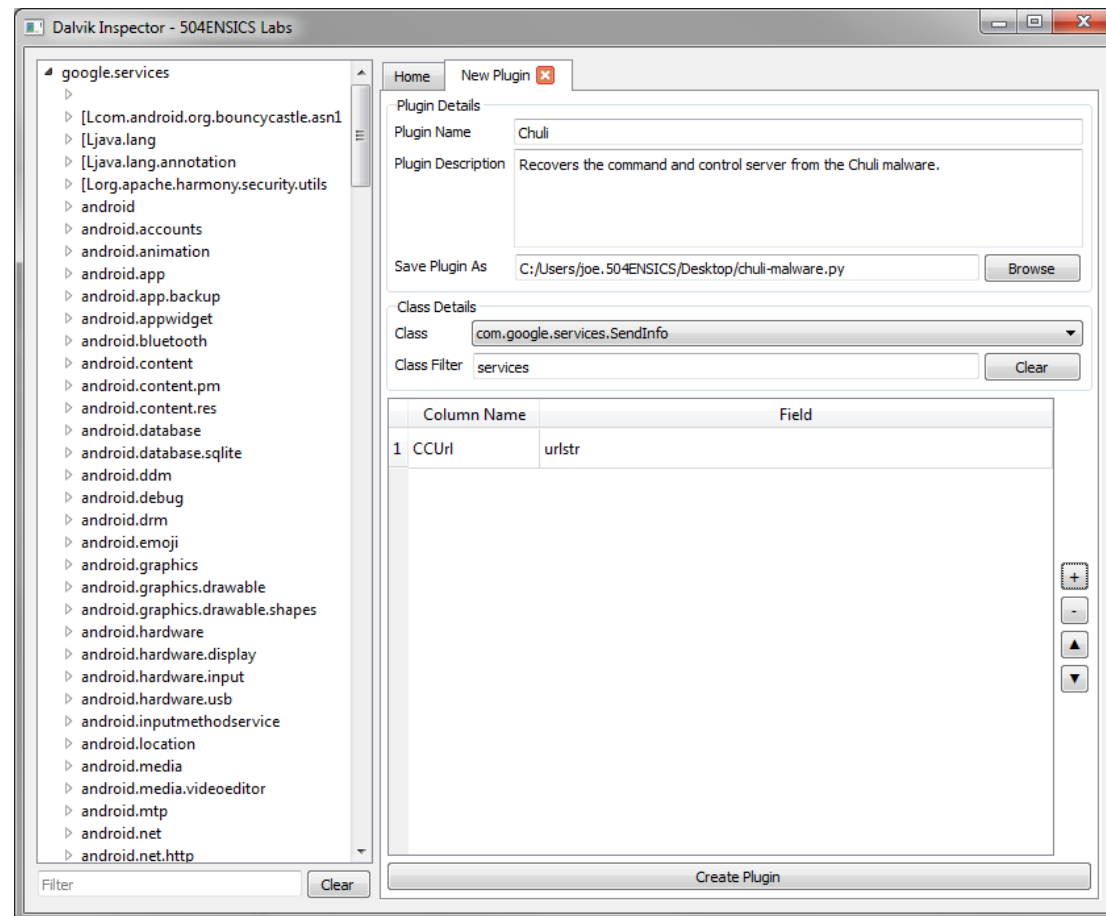
Plugin Creation



Plugin Creation



Plugin Creation



Plugin Creation

```
C:\Users\joe.504ENSICS\Desktop\chuli-malware.py - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
chuli-malware.py [X]
1
2 # Volatility
3 #
4 # This program is free software; you can redistribute it and/or modify
5 # it under the terms of the GNU General Public License as published by
6 # the Free Software Foundation; either version 2 of the License, or (at
7 # your option) any later version.
8 #
9 # This program is distributed in the hope that it will be useful, but
10 # WITHOUT ANY WARRANTY; without even the implied warranty of
11 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
12 # General Public License for more details.
13 #
14 # You should have received a copy of the GNU General Public License
15 # along with this program; if not, write to the Free Software
16 # Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
17
18 import volatility.plugins.linux.common as linux_common
19 import volatility.plugins.dalvik.db_common as db_common
20
21 class chuli(linux_common.AbstractLinuxCommand):
22     """Recovers the command and control server from the Chuli malware."""
23
24     def __init__(self, config, *args, **kwargs):
25         linux_common.AbstractLinuxCommand.__init__(self, config, *args, **kwargs)
26         config.add_option('DATABASE_PATH', short_option = 'b', default = None, help = 'Database of parsed objects and classes', action = 'store', type = 'str')
27         config.add_option('APP_PID', short_option = 'a', default = None, help = 'PID of app', action = 'store', type = 'int')
28
29     def calculate(self):
30         linux_common.set_plugin_members(self)
31
32         self.db_ops = db_common.db_operations(self._config.DATABASE_PATH, self._config.APP_PID)
33
34         yield self.db_ops.get_records("Lcom/google/services/SendInfo;", [ "urlstr",])
35
36     def render_text(self, outfd, data):
37         self.table_header(outfd, [("Application", "30"),
38                                  ("PID", "15"),
39                                  ("CCU1", "50"),])
40
41         for record in data:
42             for ( a0,) in record:
43                 self.table_row(outfd, self.db_ops.app_name, self.db_ops.app_pid, a0,)
```

Python file length: 2033 lines: 42 Ln: 42 Col: 91 Sel: 0|0 Dos\Windows ANSI as UTF-8 INS

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