Bro Intrusion Detection System

Masato Anzai
N12725403
OSS Assessment

Table of Content

Background

Bro IDS is an open source Intrusion Detection System that is being employed in major corporations world wide. The IDS is very flexible and stable for production at all deployment scales.

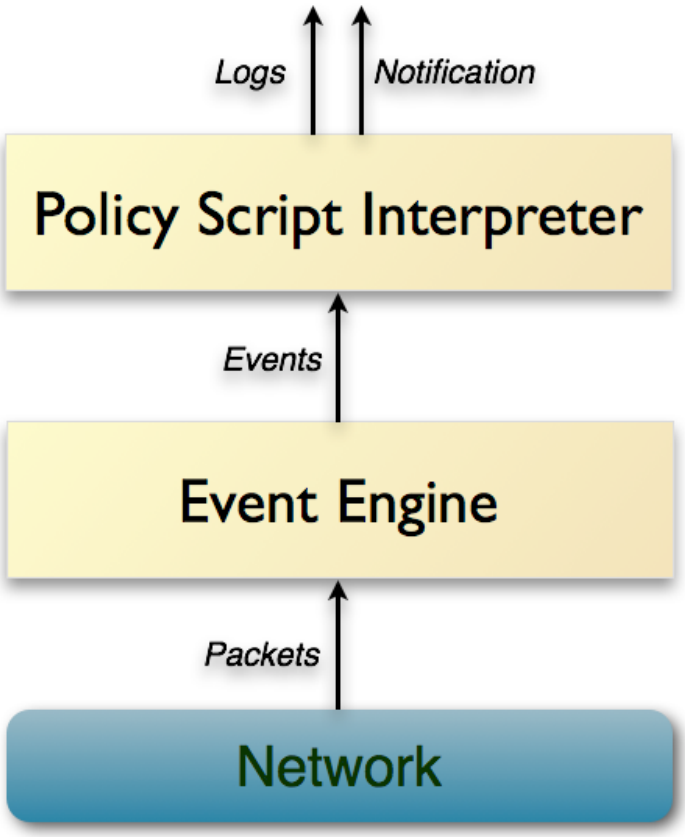
It's unique ability to apply customizable scripts makes the Bro IDS configurable to create real time alerts, execute arbitrary programs on demand, and log data for later use.

Bro's out-of-the-box installation gives you an immediate view of your network activity, including file types, software, and networked devices. You can export this data to a variety of visualization tools to provide meaningful interpretations to a broader audience.

Points of acknowledgement:

- Bro IDS integrates a fully customizable scripting language
- Create an IDS that provides immediate view of network activity
- Visualization tools for interpretation by non-technical audience
- Records log data of traffic specified by user

Architecture



Architecturally, Bros is layered into two major components. Its event engine reduces the incoming packet stream into a series of higher-level events. These events reflect network activity in policy-neutral terms, i.e., they describe what has been seen, but not why, or whether it is significant. The event however doesn't convey any further interpretation. The script interpreter is the component that handles why the event had occurred and what actions are to be executed once these events occur. More generally the script interpreter can derive any desired properties and statistics from the input traffic. The scripts can generate real time alerts and execute arbitrary external commands as needed.

Environment Settings

Operating System:
- Mac OS X El Capitan 10.11.2

Installing Bro IDS

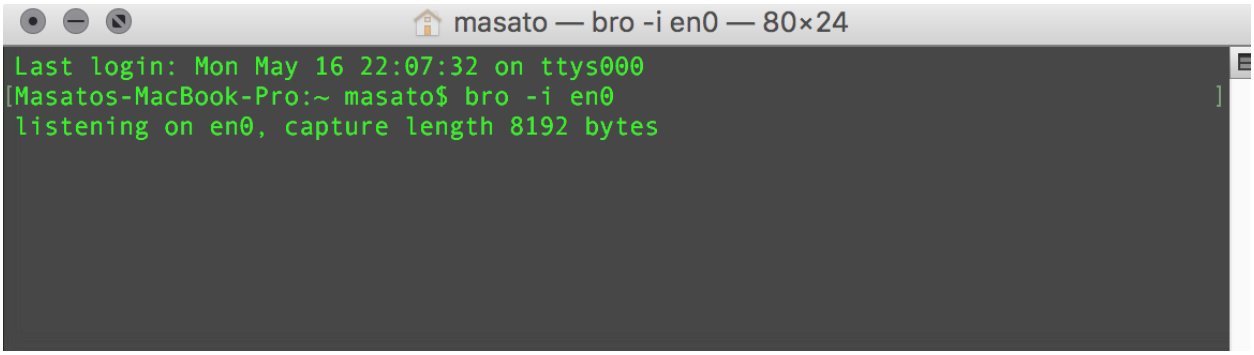
Installing Dependencies:
- Used MacPorts to install cmake, swig, swig-python
- Installed Xcode Command Line Tools for extra dependencies required
- Optional Dependencies not installed:
  - LibGeoIP
  - sendmail
  - curl
  - gperftools
  - ipsumdump

Installing Bro IDS:
- Downloaded full source code from git revision control
- “git clone —recursive git git://git.bro.org/bro” ran in terminal
- Switched directory to the bro folder and ran the configure file
- “./configure”
- Ran the make file and installed
- “make” —> “make install”

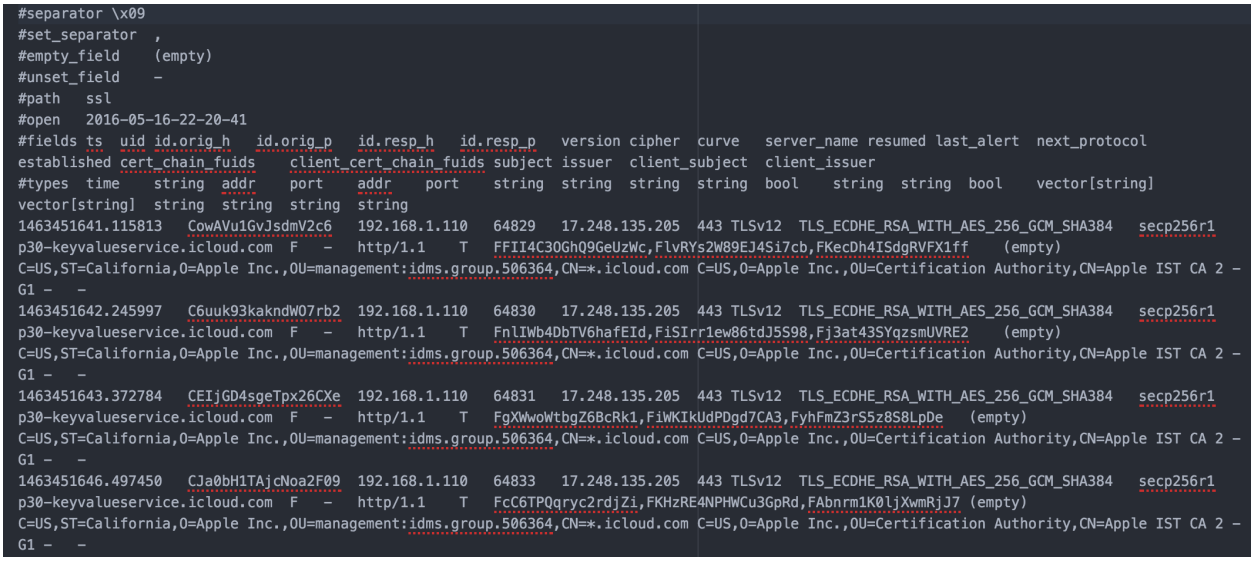
Testing Bro IDS

Monitoring Live Traffic:

- Analyzing live traffic from Bro
- “bro -i en0”
- We can add specific scripts to the command



- The logs are stored in the CD of the command
- A sample of ssl logs named default as ssl.log



```
#separator \x09
#set_separator  ,
#empty_field    (empty)
#unset_field    -
#path   ssl
#open   2016-05-16-22-20-41
#fields ts  uid id.orig_h   id.orig_p   id.resp_h   id.resp_p   version cipher  curve   server_name resumed last_alert  next_protocol
established cert_chain_fuids    client_cert_chain_fuids subject issuer  client_subject  client_issuer
#types  time    string  addr    port    addr    port    string  string  string  string  bool    string  string  bool    vector[string]
vector[string]  string  string  string  string
1463451641.115813   CowAVu1GvJsdmV2c6   192.168.1.110   64829   17.248.135.205  443 TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   secp256r1
p30-keyvalueservice.icloud.com  F   -   http/1.1    T   FFII4C3OGhQ9GeUzWc,FlvRYs2W89EJ4Si7cb,FKecDh4ISdgRVFX1ff    (empty)
C=US,ST=California,O=Apple Inc.,OU=management:idms.group.506364,CN=*.icloud.com C=US,O=Apple Inc.,OU=Certification Authority,CN=Apple IST CA 2 -
G1 -    -
1463451642.245997   C6uuk93kakndWO7rb2  192.168.1.110   64830   17.248.135.205  443 TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   secp256r1
p30-keyvalueservice.icloud.com  F   -   http/1.1    T   FnlIWb4DbTV6hafEId,FiSIrr1ew86tdJ5S98,Fj3at43SYqzsmUVRE2    (empty)
C=US,ST=California,O=Apple Inc.,OU=management:idms.group.506364,CN=*.icloud.com C=US,O=Apple Inc.,OU=Certification Authority,CN=Apple IST CA 2 -
G1 -    -
1463451643.372784   CEIjGD4sgeTpx26CXe  192.168.1.110   64831   17.248.135.205  443 TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   secp256r1
p30-keyvalueservice.icloud.com  F   -   http/1.1    T   FgXWwoWtbgZ6BcRk1,FiWKIkUdPDgd7CA3,FyhFmZ3rS5z8S8LpDe   (empty)
C=US,ST=California,O=Apple Inc.,OU=management:idms.group.506364,CN=*.icloud.com C=US,O=Apple Inc.,OU=Certification Authority,CN=Apple IST CA 2 -
G1 -    -
1463451646.497450   CJa0bH1TAjcNoa2F09  192.168.1.110   64833   17.248.135.205  443 TLSv12  TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384   secp256r1
p30-keyvalueservice.icloud.com  F   -   http/1.1    T   FcC6TPQqryc2rdjZi,FKHzRE4NPHWCu3GpRd,FAbnrm1K0ljXwmRjJ7 (empty)
C=US,ST=California,O=Apple Inc.,OU=management:idms.group.506364,CN=*.icloud.com C=US,O=Apple Inc.,OU=Certification Authority,CN=Apple IST CA 2 -
G1 -    -
```

Changing the Configurations

Changing Directory:

- Path of working directory = “/opt/local/var/macports/sources/rsync.macports.org/release/tarballs/ports/net/bro”
- Set right interface to monitor in “/etc/node.cfg”
- Set network configuration “/etc/networks.cfg”

Now start the BroControl shell like:

```
broctl
```

Since this is the first-time use of the shell, perform an initial installation of the BroControl configuration:

```
[BroControl] > install
```

Then start up a Bro instance:

```
[BroControl] > start
```

- After configuring the Bro IDS, we can start running some custom scripts

Custom Scripting

Creating a custom script:

Bro includes an event-driven scripting language that provides the primary means for an organization to extend and customize Bro's functionality. Almost all output that is created by the Bro IDS is developed through the Bro Scripts. Bro is essentially the entity behind the scenes and the client is able to create a script for the IDS. Bro scripts effectively notify Bro that should there be an event of a type we define, then let us have the information about the connection so we can perform some function on it. For example, the ssl.log file is generated by a Bro script that walks the entire certificate chain and issues notifications if any of the steps along the certificate chain are invalid. This entire process is setup by telling Bro that should it see a server or client issue an SSL HELLO message, we want to know about the information about that connection.

Classic Hello World Example:

```
event bro_init()
        {
        print "Hello, World!";
        }

event bro_done()
        {
        print "Goodbye, World!";
        }
```

Primitive Datatypes Example:

```
event bro_init()
        {
        local x : string = "two";
        local y : int = 10000000000000000000000000000000000000000000000000;
        print "y is a large int:", y;
        print "x is a short string:", x;

        #pattern matching
        print /one|two|three/ == "two";  # T
        print /one|two|three/ == "ones"; # F (exact matching)
        print /one|two|three/ in "ones"; # T (embedded matching)
```

```
        print /[123].*/ == "2 two";  # T
        print /[123].*/ == "4 four"; # F
        }
```

SSH Connection Example:

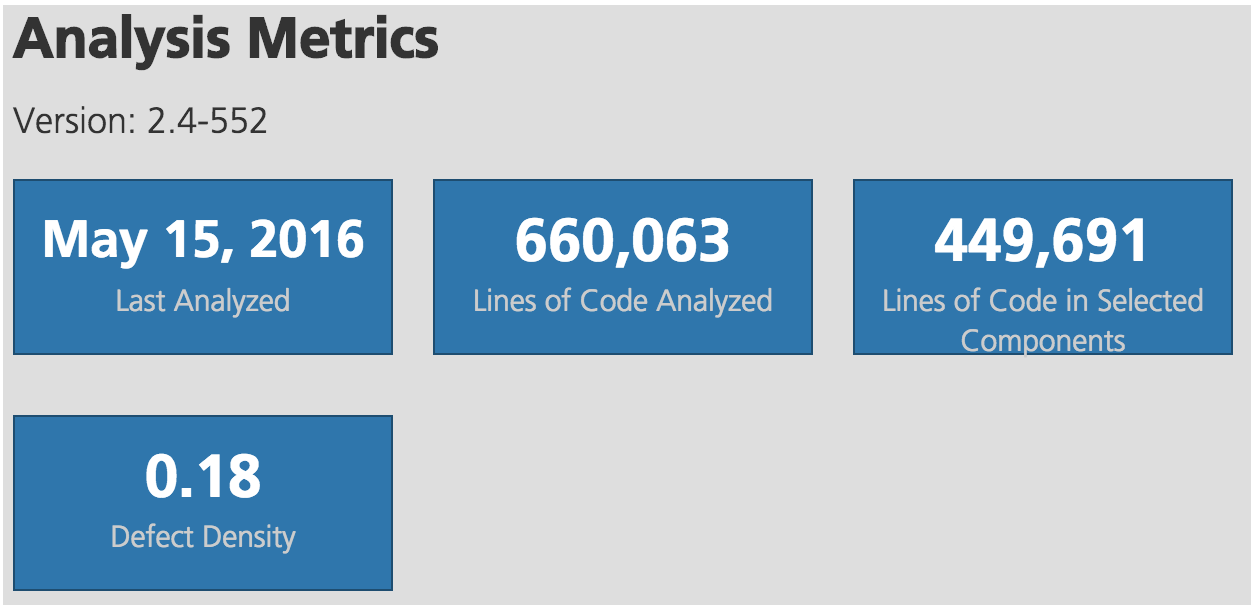
```
global ssh_connections: count = 0;
global ssh_connections_by_host: table[addr] of count &default=0;

event SSH::log_ssh(rec: SSH::Info) {
   ++ssh_connections;
   ++ssh_connections_by_host[rec$id$orig_h];
}

event bro_done() {
   print fmt("Saw %d ssh connections", ssh_connections);
   for (src in ssh_connections_by_host) {
      print fmt("%s made %d connections", src, ssh_connections_by_host[src]);
   }
}
```

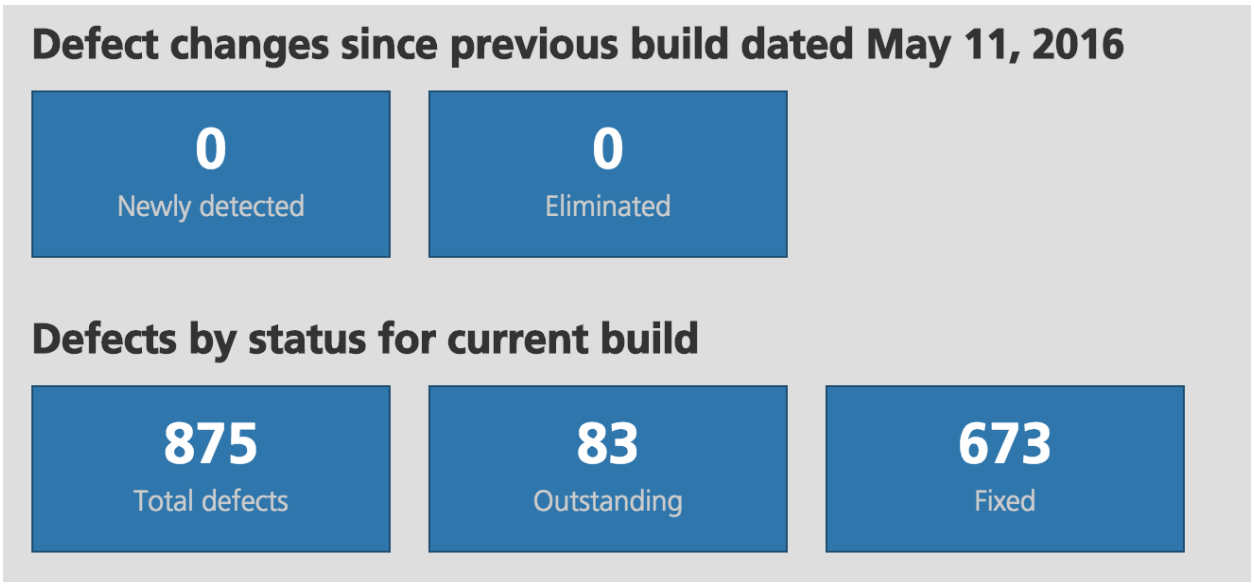
Test Overview

In order to fully test this IDS program I will analyze the code and application as a whole through static code analysis (white box testing) and black box testing. By using these methods, I can attack the program from many angles increasing the potential of finding a valid vulnerability. Static code analysis will be done by a lint program which is a static code analyzer to find potential faults in code by giving a thorough review of the code. Black box testing refers to the test for extreme output errors through certain inputs without looking at the code itself.

Static Code Analysis

Using an online community driven code analyzer called Coverity I got information on the code analysis.



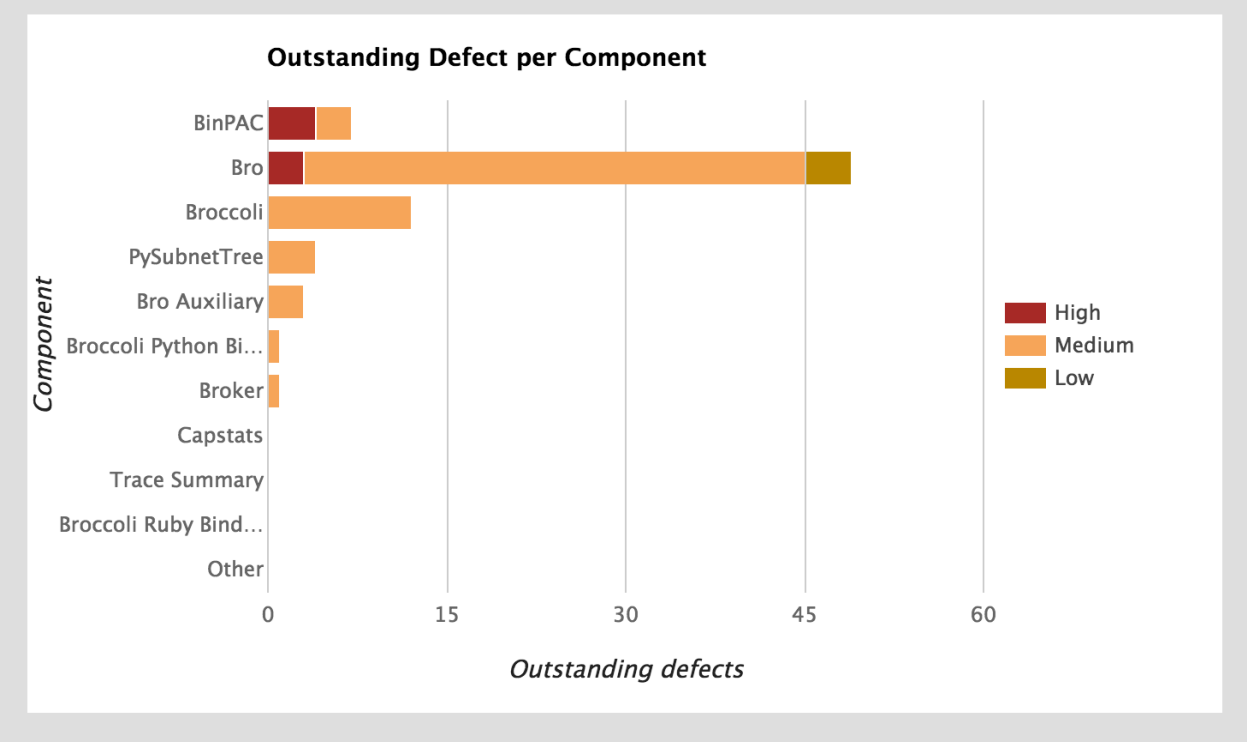


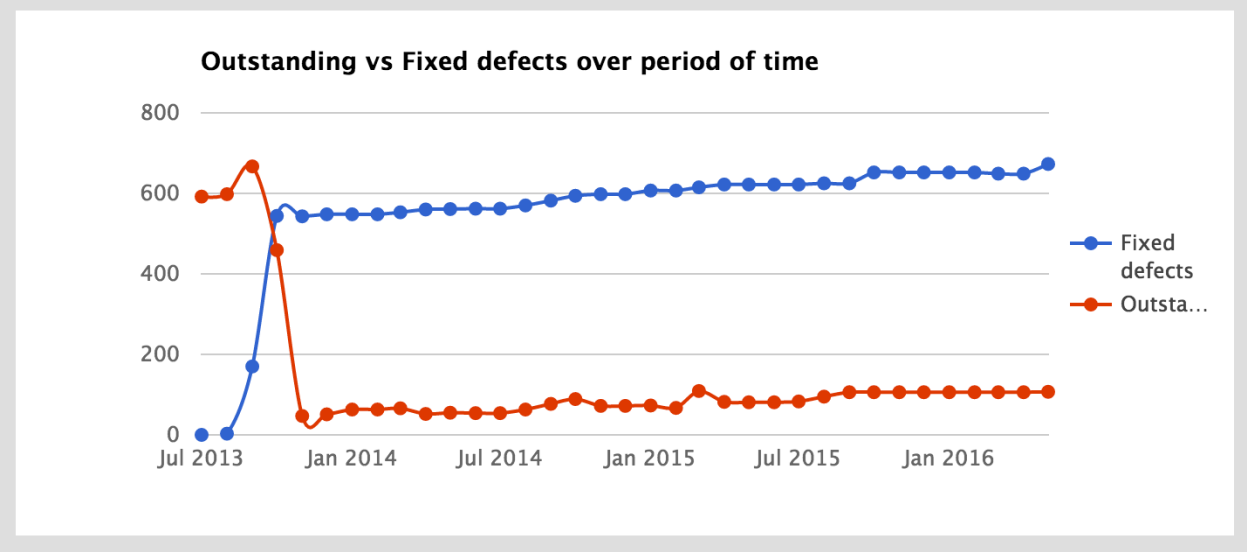
Analysis Metrics per Components

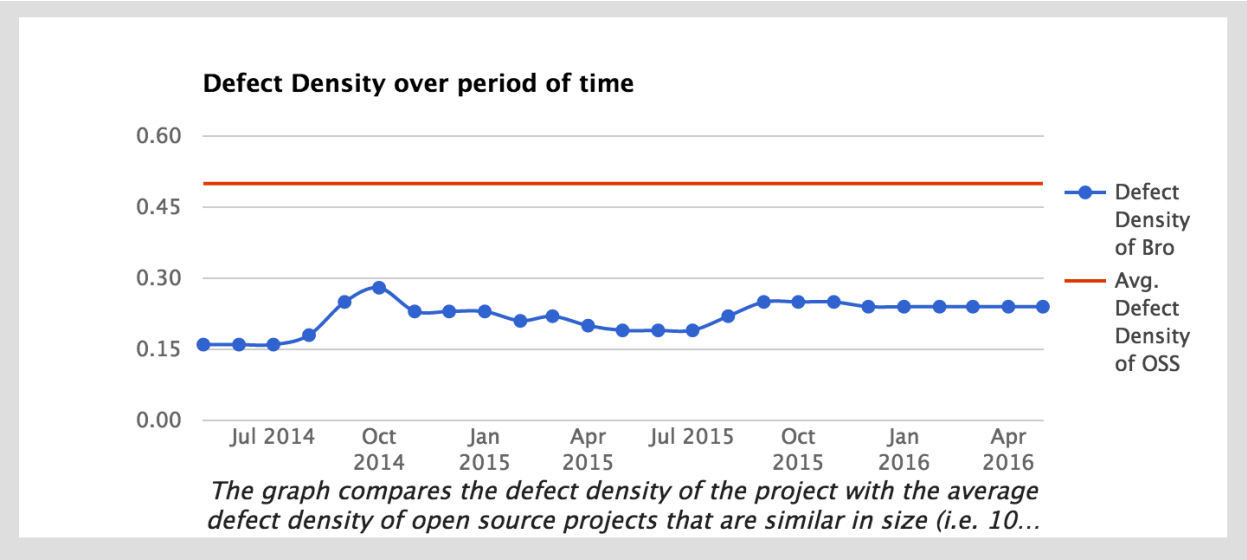
Component Name	Pattern	Ignore	Line of Code	Defect density
SQLite	.*/sqlite3.*	Yes	210,372	N/A
Broker	.*/aux/broker/.*	No	14,594	0.21
BinPAC	.*/aux/binpac/.*	No	15,559	0.71
Bro Auxiliary	.*/aux/bro-aux/.*	No	1,084	2.77
Broccoli	.*/aux/broccoli/src/.*	No	12,899	0.93
Broccoli Python Bindings	.*/aux/broccoli/bindings/broccoli-python/.*	No	3,501	0.29
Broccoli Ruby Bindings	.*/aux/broccoli/bindings/broccoli-ruby/.*	No	0	N/A
PySubnetTree	.*/aux/broctl/aux/pysubnettree/.*	No	5,523	0.72
Trace Summary	.*/aux/broctl/aux/trace-summary/.*	No	0	N/A
Third Party	.*/src/3rdparty/.*	Yes	0	N/A
libmagic	.*/libmagic-prefix/.*	Yes	0	N/A
Bro	.*	No	396,031	0.14
Other	.*	No	0	N/A

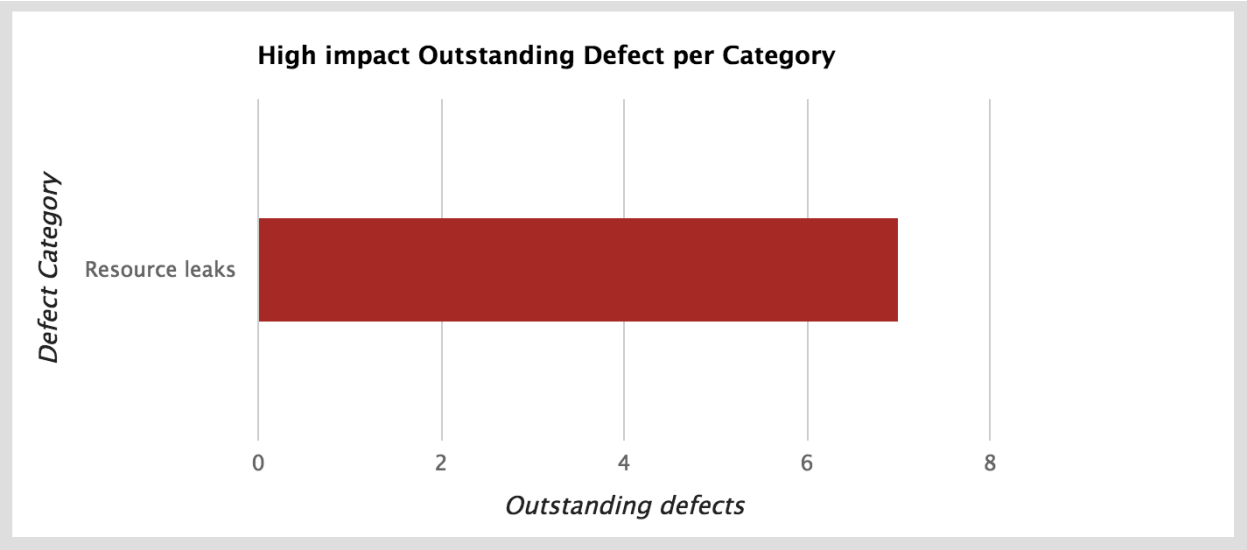
CWE Top 25 defects

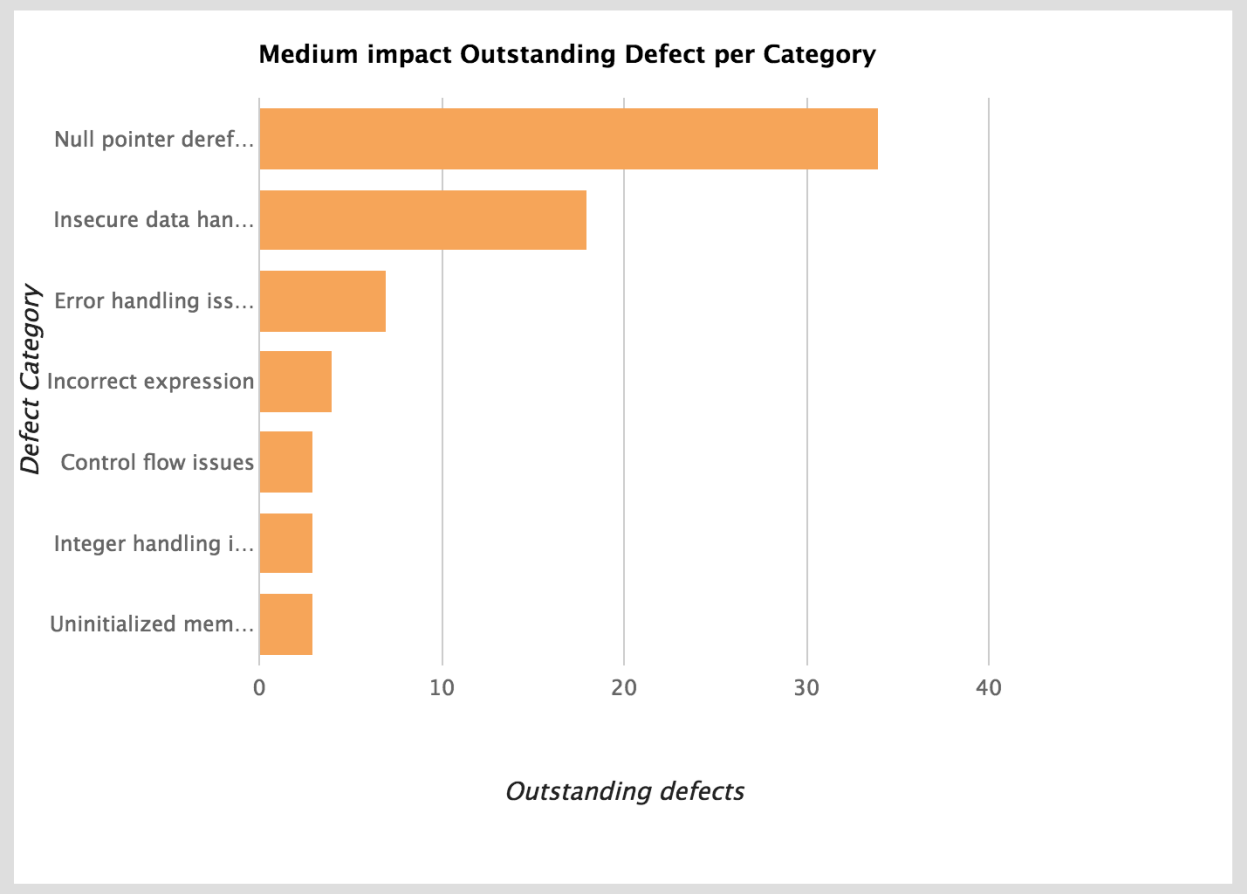
ID	CWE-Name	Number of Defects
190	Integer Overflow or Wraparound	2
676	Use of Potentially Dangerous Function	1


Outstanding Defect per Component
BinPAC
Bro
Broccoli
PySubnetTree
Bro Auxiliary
Broccoli Python Bi...
Broker
Capstats
Trace Summary
Broccoli Ruby Bind...
Other
Component
0
15
30
45
60
Outstanding defects
High
Medium
Low


Outstanding vs Fixed defects over period of time
800
600
400
200
0
Jul 2013
Jan 2014
Jul 2014
Jan 2015
Jul 2015
Jan 2016
Fixed defects
Outsta...


Defect Density over period of time
0.60
0.45
0.30
0.15
0.00
Jul 2014
Oct 2014
Jan 2015
Apr 2015
Jul 2015
Oct 2015
Jan 2016
Apr 2016
Defect Density of Bro
Avg. Defect Density of OSS
The graph compares the defect density of the project with the average defect density of open source projects that are similar in size (i.e. 10...


High impact Outstanding Defect per Category
Defect Category
Resource leaks
0
2
4
6
8
Outstanding defects



With the information provide there are several problems that were found in the code of Bro IDS. Considering it is an intrusion detection system, one would assume that the security of the application itself is of the highest quality. There are 83 outstanding defects within the source code and one that is alarming is the “Resource Leak” problem within the main BinPac application. This is the code that runs the user created scripts and is responsible for the implementation of the scripts.

We also see that there are multiple outstanding medium level defects, but this is considerably low.

The history of the code implicates that there has been major code review done around 2013-2014 when there was a massive change in the amount of potential problems in the code.

Black Box Testing

What I find curious about this Bro IDS program is that you can potentially create your own script. Given that the attacker is familiar with the scripts within Bro, it could be potentially harmful if the attacker was able to gain access to your files within the scripts. I wanted to test the functionality of the script in order to gain access to the file context.

I fed the Bro IDS program a very simple script that pulls the text out of a file name.

```
global Passwords: table[addr] of Val = table();

Input::add_table([$source=“passwords.file", $name="password", $idx=Idx, $val=Val, $destination=password]);
Input::remove("password");
```

I took the idea from the input function that’s provided for users to access a blacklist domain list.

```
event Input::update_finished(name: string, source: string) {
	# now all data is in the table
	print password;
}
```

Afterwards I can easily find the values that are written within the file.

Understanding easy commands within Bro allows an attacker to access a new attack vector that was potentially unavailable.

Application of Static Code Review

One large vulnerability that I saw within the code review is that the BinPAC, a high level language for protocol parsing and C++ code generation had Resource Leaking. In fact if you alter some of the code provided within the BinPAC README you can create a C++ code out of BinPAC that has an out of bounds checking problem. This could potentially let hackers send remote protocol headers that trigger the out of bounds pointers.

```
 // PAC type record. This is stored in myProtocol.pac
 type myProtocol = record {
    data:uint8;
 };

// The binpac utility generates this C++ class to parse myProtocol
 class myProtocol {
 public:
    myProtocol();
    ~myProtocol();
    int Parse(const_byteptr const t_begin_of_data, const_byteptr const t_end_of_data);
    uint8 data() const  { return data_; }
 protected:
    uint8 data_;
 };

// The resulting binpac utility generated Parse() function
myProtocol::Parse(const_byteptr const t_begin_of_data, const_byteptr const
t_end_of_data) {
  // Assign data_ from beginning of protocol bytes
  data_ = *((uint8 const *) ((t_begin_of_data)));
  …
}
```

Suggestions

Boundaries for user scripts:
- It may be smart to check for memory corruption
- Check for buffer overflows
- Limit the potential application of scripts
- Create a sandbox
- White list more functions

Fix the potential Resource Leak:
- Implement better boundary checks
- Create a safer way to generate code out of the high level BinPAC language

References:

Bruce Schneier: Crypto-Gram June 15, 2014: "The NSA is Not Made Of Magic", likening the NSA tool "X-KEYSTORE" to "Bro plus memory".

"Bro: A System for Detecting Network Intruders in Real-Time." The ICSI Networking and Security Group. Web. 17 May 2016.

"Installing Bro." — Bro 2.4.1 Documentation. Web. 17 May 2016.

"Introduction." — Bro 2.4.1 Documentation. Web. 17 May 2016.

"Coverity Scan." - Static Analysis. Web. 17 May 2016.