Bro vs Suricata

Two Approaches to Network Security Monitoring

Christian Kreibich
christian@corelight.com
@ckreibich

Your speaker

ICSI
INTERNATIONAL COMPUTER SCIENCE INSTITUTE

lastline

corelight

Part 1

Background on Bro

(✓)

Already covered yesterday

Part 2

Background on Suricata

October 16, 2008 (LAFAYETTE, Ind.) – The Open Information Security Foundation (**OISF**, www.openinfosecfoundation.org) is proud to announce its formation, made possible by a grant from the U.S. **Department of Homeland Security** (DHS). The OISF has been chartered and funded by DHS to **build a next-generation intrusion detection and prevention engine**. This project will consider every new and existing technology, concept and idea to build a completely open source licensed engine. Development will be funded by DHS, and the end product will be made available to any user or organization.

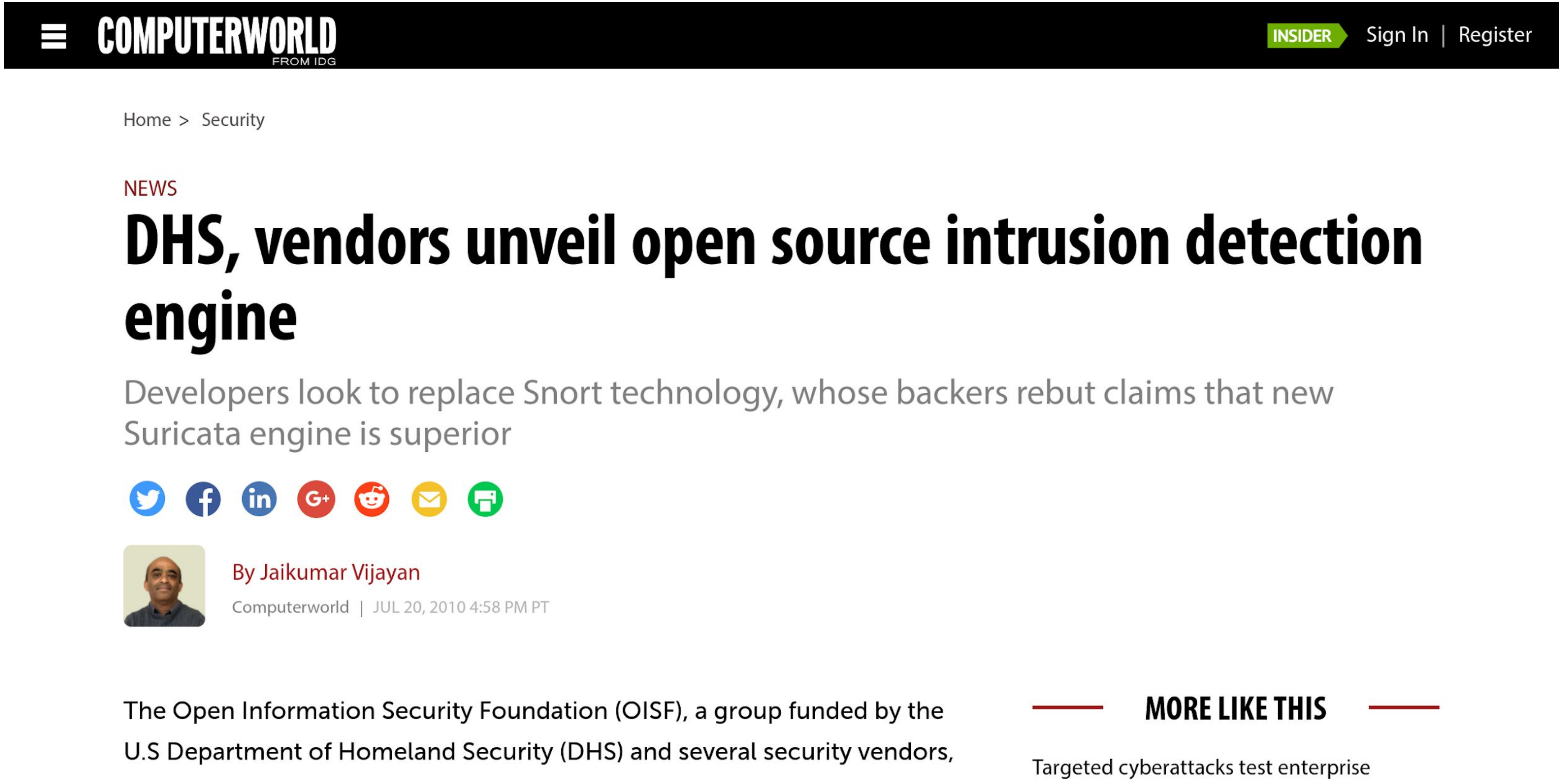
```
From: jonkman at jonkmans.com (Matt Jonkman)
To: discussion@lists.openinfosecfoundation.org
Date: Thu, 16 Oct 2008 21:00:32 -0400
Subject: [Discussion] Features

So here's my wish list:

1. Native multithreading.

2. IP Reputation Sharing

3. Native ipv6

4. Native Hardware acceleration support

5. Scoring
```

A few dozen brainstorm emails later ...

From: hall.692 at osu.edu (**Seth Hall**)
To: discussion@lists.openinfosecfoundation.org
Date: Sun, 19 Oct 2008 00:25:34 -0400
Subject: [Discussion] **I think everyone is describing Bro**

Sorry, I just joined the list so I'm going to be doing some odd quoting from the list archive :) I do want to point out too, that I'm not writing this email to downplay OISFs goals but rather to **hopefully guide OISF toward improving an existing opensource project (Bro** - http://www.bro-ids.org/) that already does much of what is being discussed on this list.

… but 1.5 years later …



COMPUTERWORLD FROM IDG

INSIDER Sign In | Register

Home > Security

NEWS

DHS, vendors unveil open source intrusion detection engine

Developers look to replace Snort technology, whose backers rebut claims that new Suricata engine is superior

By Jaikumar Vijayan

Computerworld | JUL 20, 2010 4:58 PM PT

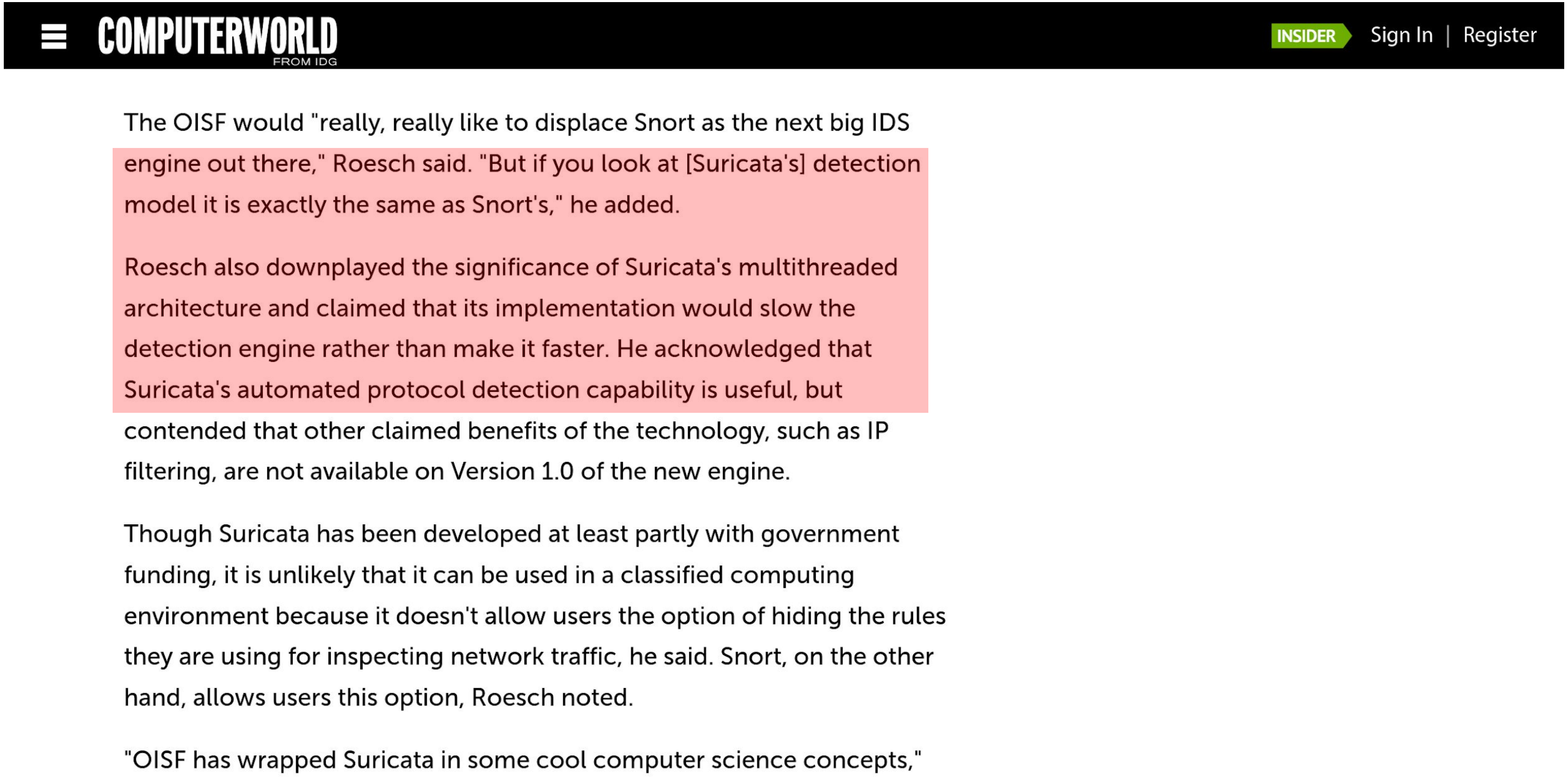
The Open Information Security Foundation (OISF), a group funded by the U.S Department of Homeland Security (DHS) and several security vendors,

MORE LIKE THIS

Targeted cyberattacks test enterprise

Suricata == Snort++

(2010)

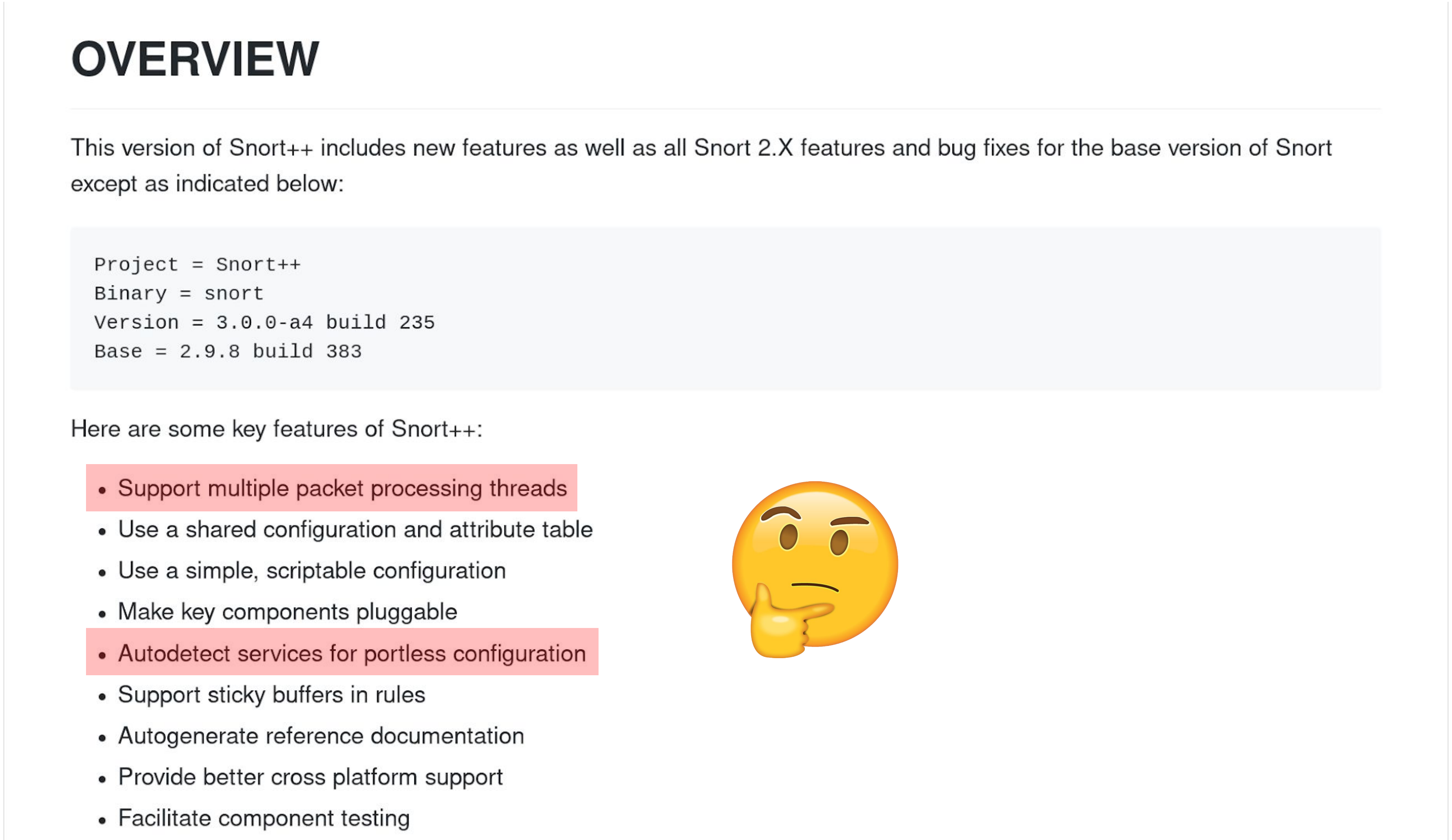


The OISF would "really, really like to displace Snort as the next big IDS engine out there," Roesch said. "But if you look at [Suricata's] detection model it is exactly the same as Snort's," he added.

Roesch also downplayed the significance of Suricata's multithreaded architecture and claimed that its implementation would slow the detection engine rather than make it faster. He acknowledged that Suricata's automated protocol detection capability is useful, but contended that other claimed benefits of the technology, such as IP filtering, are not available on Version 1.0 of the new engine.

Though Suricata has been developed at least partly with government funding, it is unlikely that it can be used in a classified computing environment because it doesn't allow users the option of hiding the rules they are using for inspecting network traffic, he said. Snort, on the other hand, allows users this option, Roesch noted.

"OISF has wrapped Suricata in some cool computer science concepts," From https://github.com/snort3/snort3/blob/master/README.md …



OVERVIEW

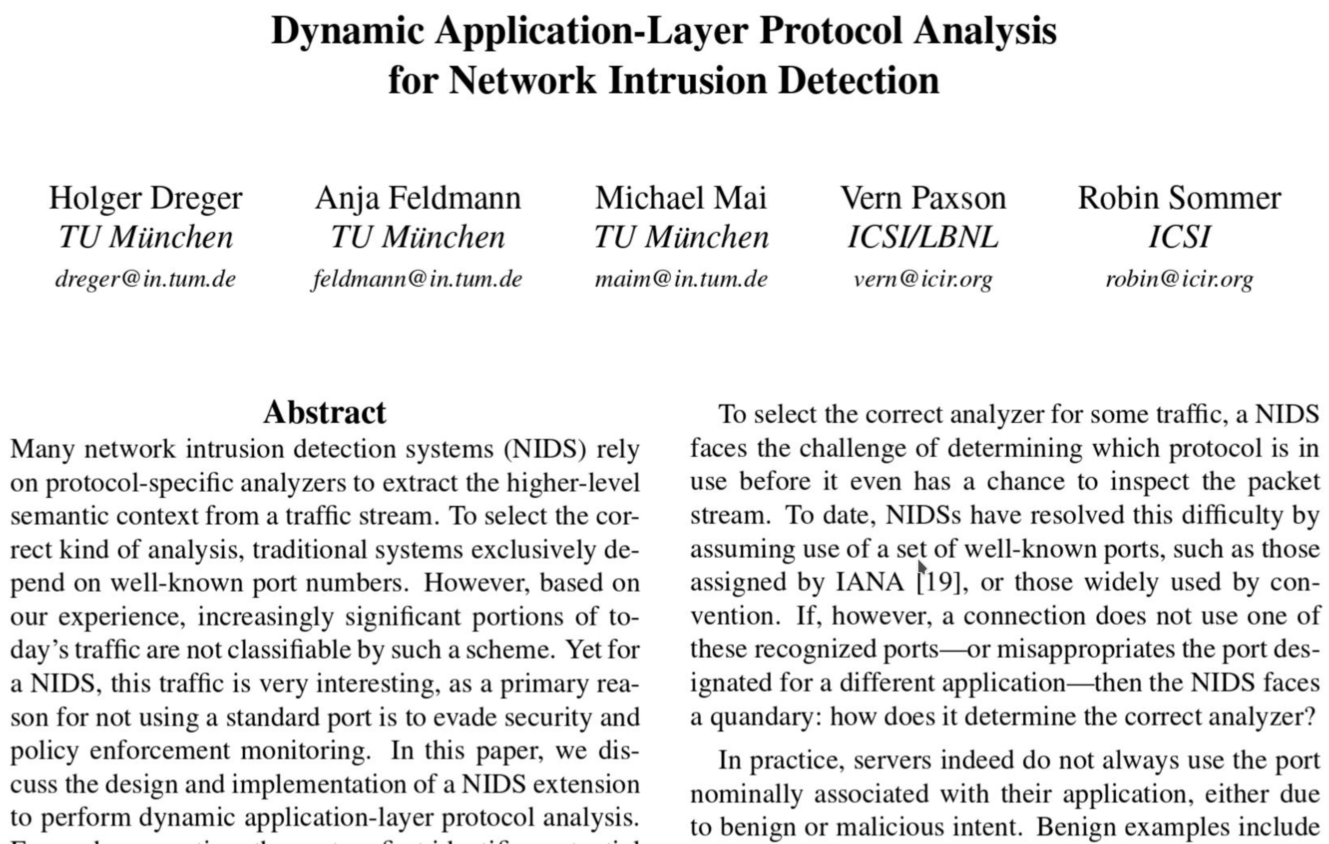
This version of Snort++ includes new features as well as all Snort 2.X features and bug fixes for the base version of Snort except as indicated below:

```
Project = Snort++
Binary = snort
Version = 3.0.0-a4 build 235
Base = 2.9.8 build 383
```

Here are some key features of Snort++:

- Support multiple packet processing threads
- Use a shared configuration and attribute table
- Use a simple, scriptable configuration
- Make key components pluggable
- Autodetect services for portless configuration
- Support sticky buffers in rules
- Autogenerate reference documentation
- Provide better cross platform support
- Facilitate component testing

USENIX Security ... 2006



Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection

Holger Dreger
TU München
dreger@in.tum.de

Anja Feldmann
TU München
feldmann@in.tum.de

Michael Mai
TU München
maim@in.tum.de

Vern Paxson
ICSI/LBNL
vern@icir.org

Robin Sommer
ICSI
robin@icir.org

Abstract

Many network intrusion detection systems (NIDS) rely on protocol-specific analyzers to extract the higher-level semantic context from a traffic stream. To select the correct kind of analysis, traditional systems exclusively depend on well-known port numbers. However, based on our experience, increasingly significant portions of today's traffic are not classifiable by such a scheme. Yet for a NIDS, this traffic is very interesting, as a primary reason for not using a standard port is to evade security and policy enforcement monitoring. In this paper, we discuss the design and implementation of a NIDS extension to perform dynamic application-layer protocol analysis.

To select the correct analyzer for some traffic, a NIDS faces the challenge of determining which protocol is in use before it even has a chance to inspect the packet stream. To date, NIDSs have resolved this difficulty by assuming use of a set of well-known ports, such as those assigned by IANA [19], or those widely used by convention. If, however, a connection does not use one of these recognized ports—or misappropriates the port designated for a different application—then the NIDS faces a quandary: how does it determine the correct analyzer?

In practice, servers indeed do not always use the port nominally associated with their application, either due to benign or malicious intent. Benign examples include

Suricata ~ Snort 3

(2018)

Typical signature

```
alert tcp $HOME_NET any -> $EXTERNAL_NET any
(msg:”ET TROJAN Likely Bot Nick in IRC”;
flow:established,to_server;
flowbits:isset,is_proto_irc; content:”NICK “;
pcre:”/NICK .*USA.*[0-9]{3,}/i”;
reference:url,doc.emergingthreats.net/2008124;
classtype:trojan-activity; sid:2008124; rev:2;)
```

Traditional alert log

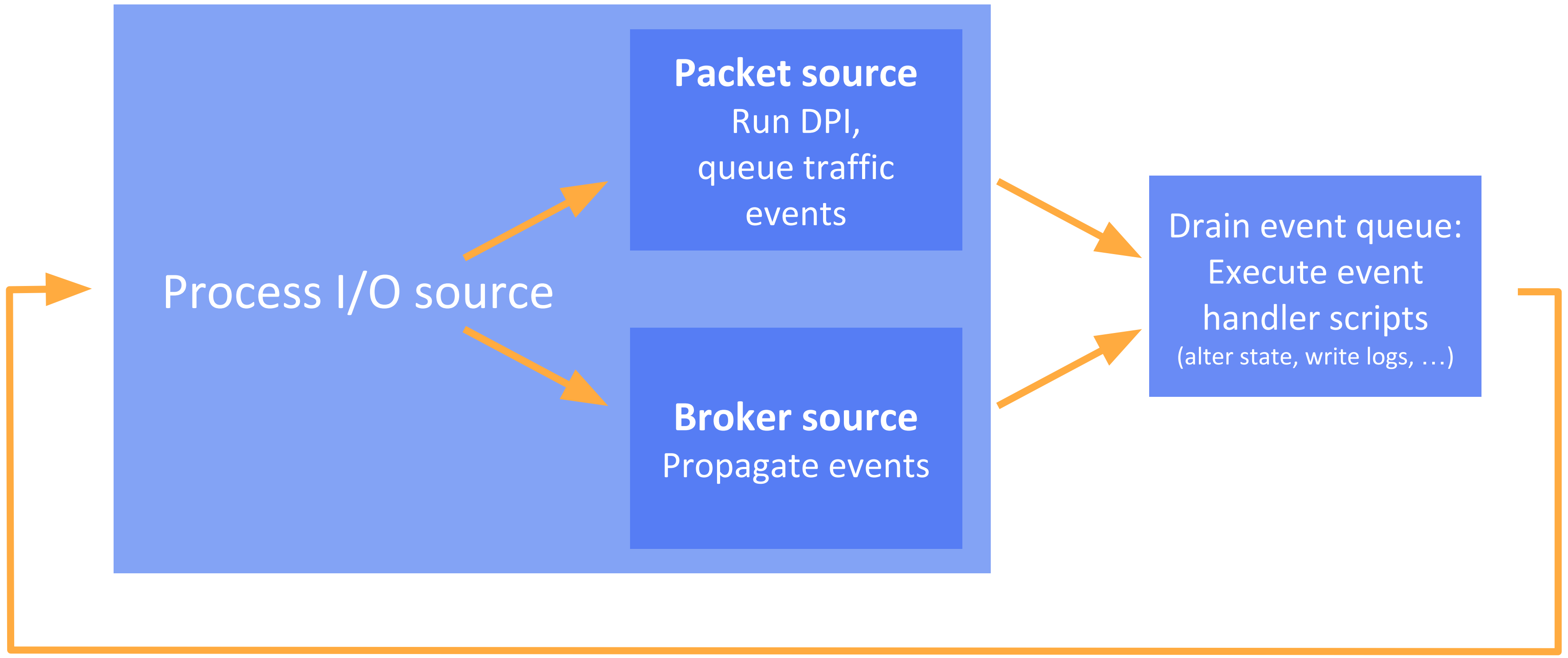
```
10/05/10-10:08:59.667372 [**] [1:2008124:2] ET
TROJAN Likely Bot Nick in IRC [**]
[Classification: Trojan Activity] [Priority: 3]
{TCP} 10.0.1.144:6984 -> 192.168.1.4:56068
```

(Today usually done via Suricata’s “EVE” JSON log)

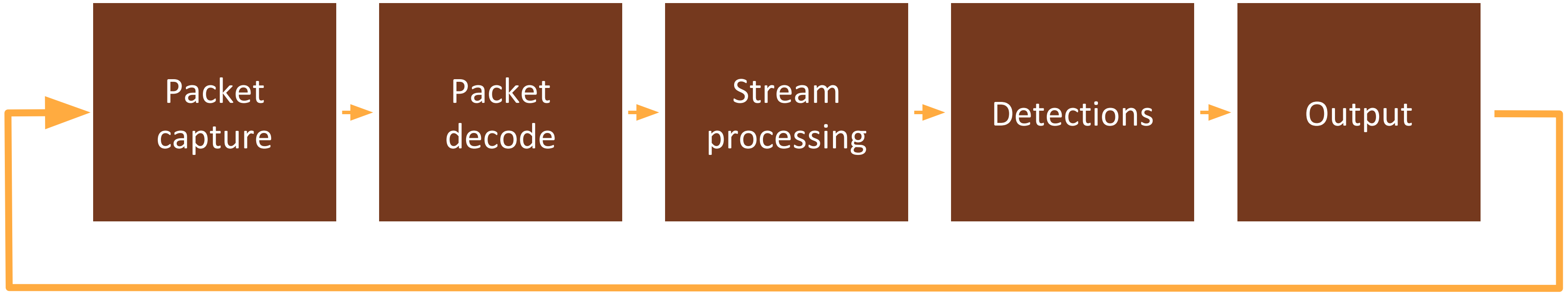
Part 3

Architecture comparison

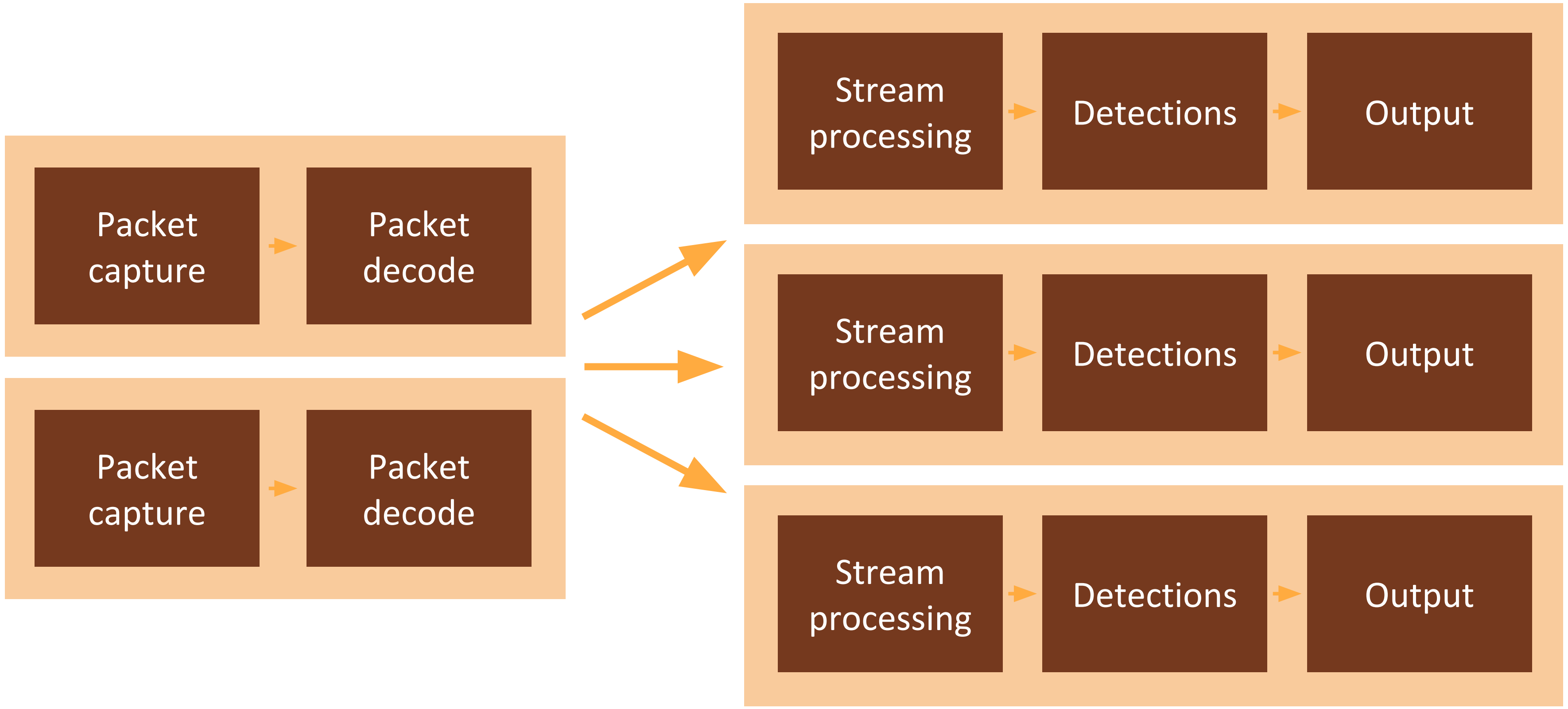
Execution flow: Bro



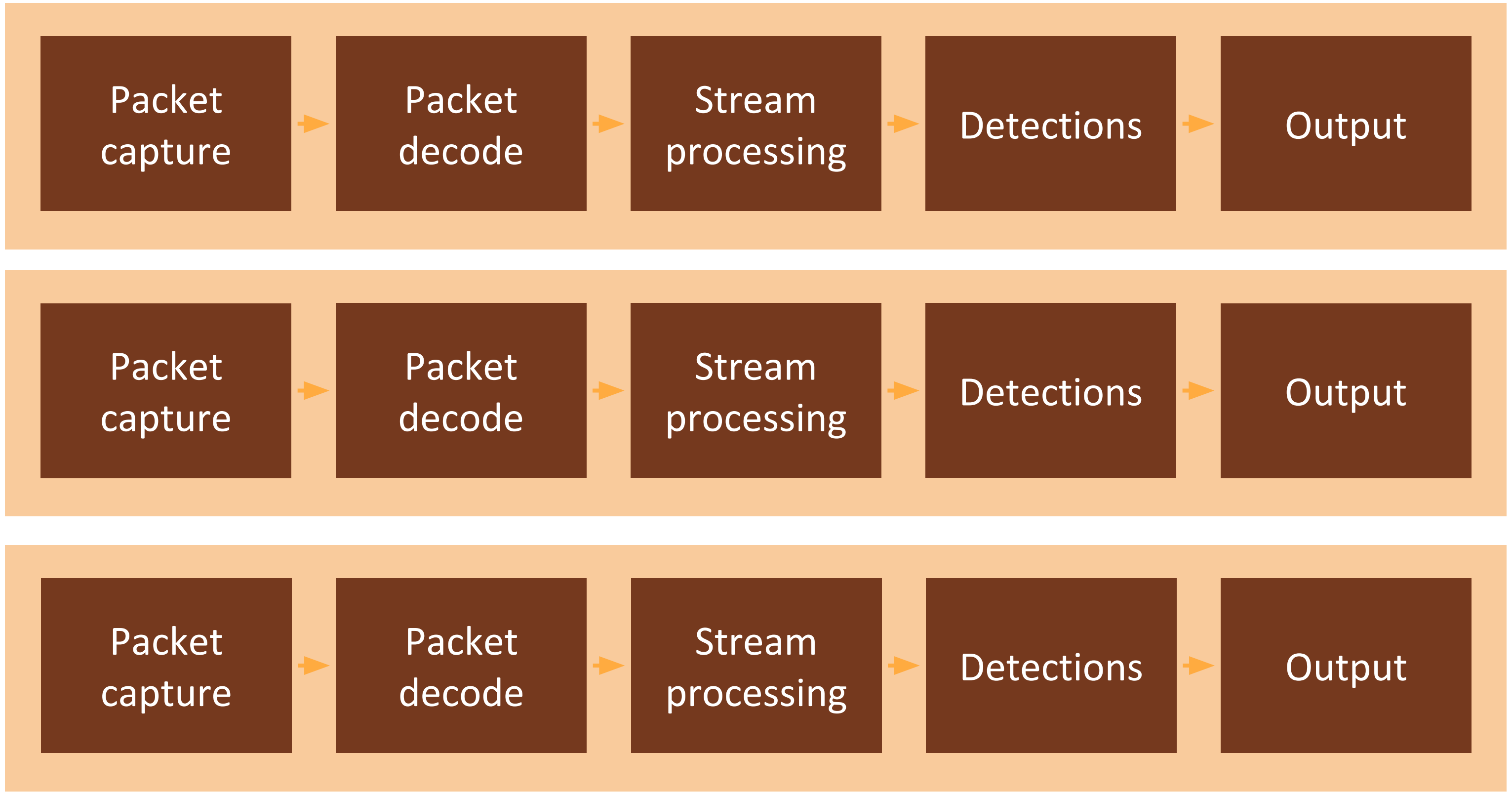
Execution flow: Suricata



Execution flow: Suricata runmodes



Execution flow: Suricata runmodes



Keeping state

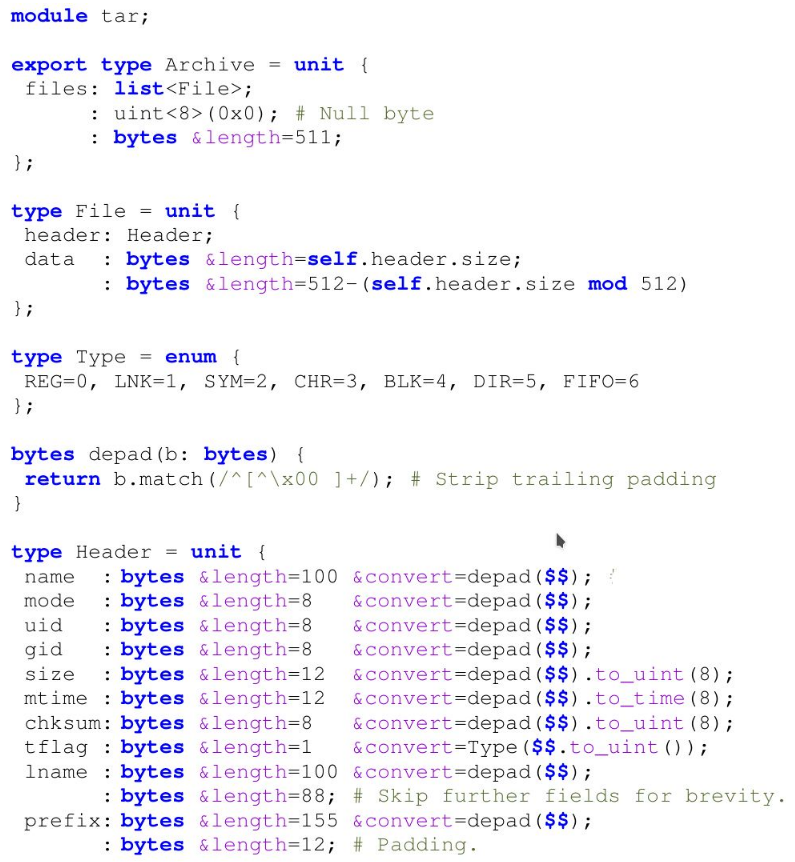
- In Bro: in scripts
  - Tables, sets, ..., with DSL convenience
  - Can be tricky if shared across cluster
- In Suricata: in rule language
  - Named bits or counter variables, per flow/host/host-pair

```
alert tcp any any -> any any (msg:"More than Five Usernames!";
content:"jonkman"; flowint: usernamecount, +, 1;
flowint:usernamecount, >, 5;)
```

Parsing traffic

- In Bro: in C++ or BinPAC
  - Phasing out BinPAC, transitioning to HILTI / Spicy
  - The latter do not depend on Bro, work w/ other tools
  - Goal: safe grammars for syntax and semantics in DSL
- In Suricata: in C or Rust
  - Experimental Rust support since 4.0
  - Parsers are Suricata-specific Rust shared libs
  - Provides safety, but separates syntax and semantics

Parser example: Spicy [ACSAC'16]



```
module tar;

export type Archive = unit {
 files: list<File>;
      : uint<8>(0x0); # Null byte
      : bytes &length=511;
};

type File = unit {
 header: Header;
 data   : bytes &length=self.header.size;
        : bytes &length=512-(self.header.size mod 512)
};

type Type = enum {
 REG=0, LNK=1, SYM=2, CHR=3, BLK=4, DIR=5, FIFO=6
};

bytes depad(b: bytes) {
 return b.match(/^[^\x00 ]+/); # Strip trailing padding
}

type Header = unit {
 name   : bytes &length=100 &convert=depad($$);
 mode   : bytes &length=8   &convert=depad($$);
 uid    : bytes &length=8   &convert=depad($$);
 gid    : bytes &length=8   &convert=depad($$);
 size   : bytes &length=12  &convert=depad($$).to_uint(8);
 mtime  : bytes &length=12  &convert=depad($$).to_time(8);
 chksum: bytes &length=8   &convert=depad($$).to_uint(8);
 tflag  : bytes &length=1   &convert=Type($$.to_uint());
 lname  : bytes &length=100 &convert=depad($$);
        : bytes &length=88; # Skip further fields for brevity.
 prefix: bytes &length=155 &convert=depad($$);
        : bytes &length=12; # Padding.
```

Parser example: Rust [SPW'17]

```
pub struct TlsServerHelloV13Contents<'a> {
    pub version: u16,
    pub random: &'a[u8],
    pub cipher: u16,

    pub ext: Option<&'a[u8]>,
}

pub fn parse_tls_server_hello_tlsv13draft18(i:&[u8])
    -> IResult<&[u8],TlsMessageHandshake>
{
    do_parse!(i,
        hv:     be_u16 >>
        random: take!(32) >>
        cipher: be_u16 >>
        ext:    opt!(length_bytes!(be_u16)) >>
        (
            TlsMessageHandshake::ServerHelloV13(
                TlsServerHelloV13Contents::new(hv,random,
                     cipher,ext)
            )
        )
    )
}
```

Extensibility

- In Bro: naturally via scripting, also plugins
  - Plugins allow native-code extensions without patching
  - Can add e.g. entire protocol analyzer
- In Suricata:
  - Monolithic architecture, usually need to patch ...
  - … unless Lua suffices: can add parsers, detectors, and loggers

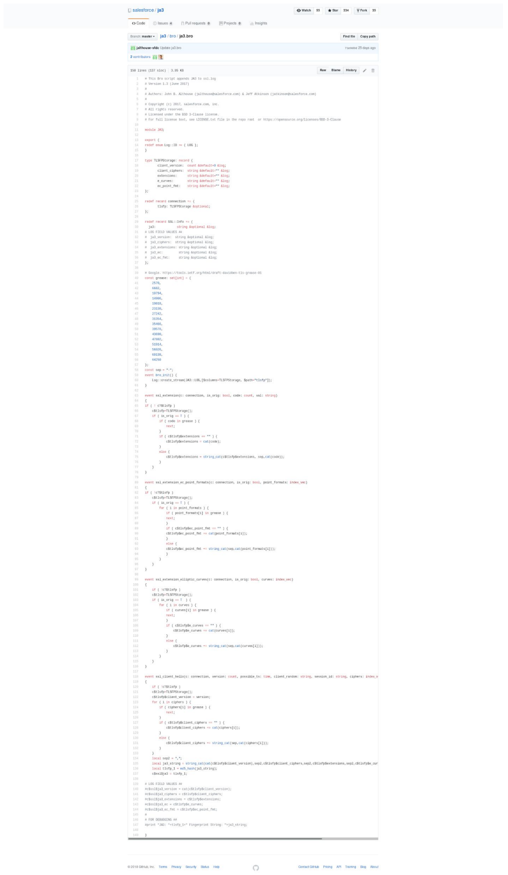
Suricata Lua script example: PDF obfuscation [NVISO]

```
tBlacklisted = {["/JavaScript"] = true}

function PDFCheckName(sInput)
    for sMatchedName in sInput:gmatch"/[a-zA-Z0-9_#]+" do
        if sMatchedName:find("#") then
            local sNormalizedName = sMatchedName:gsub("#[a-fA-F0-9][a-fA-F0-9]",
                function(hex) return string.char(tonumber(hex:sub(2), 16)) end)
            if tBlacklisted[sNormalizedName] then
                return 1
            end
        end
    end
    return 0
end

function init(args)
    return {["http.response_body"] = tostring(true)}
end

function match(args)
    return PDFCheckName(tostring(args["http.response_body"]))
end
```

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"NVISO PDF file lua";
flow:established,to_client; luajit:pdfcheckname.lua; classtype:policy-violation;
sid:1000000; rev:1;)
```

Extension example: JA3 in Bro

150-line Bro script:



salesforce / ja3

Watch 55 | Star 334 | Fork 35

Code | Issues 4 | Pull requests 0 | Projects 0 | Insights

Branch: master | ja3 / bro / ja3.bro | Find file | Copy path

jalthouse-sfdc Update ja3.bro — 71e0d0d 25 days ago

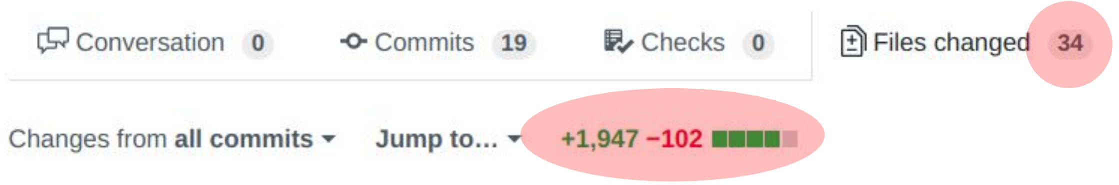
2 contributors

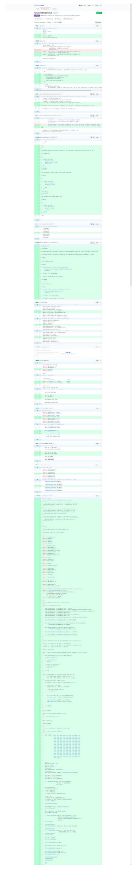
150 lines (137 sloc) | 3.95 KB — Raw | Blame | History

```
# This Bro script appends JA3 to ssl.log
# Version 1.3 (June 2017)
#
# Authors: John B. Althouse (jalthouse@salesforce.com) & Jeff Atkinson (jatkinson@salesforce.com)
#
# Copyright (c) 2017, salesforce.com, inc.
# All rights reserved.
# Licensed under the BSD 3-Clause license.
# For full license text, see LICENSE.txt file in the repo root  or https://opensource.org/licenses/BSD-3-Clause

module JA3;

export {
redef enum Log::ID += { LOG };
}

type TLSFPStorage: record {
       client_version:  count &default=0 &log;
       client_ciphers:  string &default="" &log;
       extensions:      string &default="" &log;
       e_curves:        string &default="" &log;
       ec_point_fmt:    string &default="" &log;
};

redef record connection += {
       tlsfp: TLSFPStorage &optional;
};

redef record SSL::Info += {
  ja3:            string &optional &log;
# LOG FIELD VALUES ##
#  ja3_version:  string &optional &log;
#  ja3_ciphers:  string &optional &log;
#  ja3_extensions: string &optional &log;
#  ja3_ec:         string &optional &log;
#  ja3_ec_fmt:     string &optional &log;
};

# Google. https://tools.ietf.org/html/draft-davidben-tls-grease-01
const grease: set[int] = {
    2570,
    6682,
    10794,
    14906,
    19018,
    23130,
    27242,
    31354,
    35466,
    39578,
    43690,
    47802,
    51914,
    56026,
    60138,
    64250
};
const sep = "-";
event bro_init() {
    Log::create_stream(JA3::LOG,[$columns=TLSFPStorage, $path="tlsfp"]);
}

event ssl_extension(c: connection, is_orig: bool, code: count, val: string)
{
if ( ! c?$tlsfp )
    c$tlsfp=TLSFPStorage();
    if ( is_orig == T ) {
        if ( code in grease ) {
            next;
        }
        if ( c$tlsfp$extensions == "" ) {
            c$tlsfp$extensions = cat(code);
        }
        else {
            c$tlsfp$extensions = string_cat(c$tlsfp$extensions, sep,cat(code));
        }
    }
}

event ssl_extension_ec_point_formats(c: connection, is_orig: bool, point_formats: index_vec)
{
if ( !c?$tlsfp )
    c$tlsfp=TLSFPStorage();
    if ( is_orig == T ) {
        for ( i in point_formats ) {
            if ( point_formats[i] in grease ) {
            next;
            }
            if ( c$tlsfp$ec_point_fmt == "" ) {
            c$tlsfp$ec_point_fmt += cat(point_formats[i]);
            }
            else {
            c$tlsfp$ec_point_fmt += string_cat(sep,cat(point_formats[i]));
            }
        }
    }
}

event ssl_extension_elliptic_curves(c: connection, is_orig: bool, curves: index_vec)
{
    if ( !c?$tlsfp )
    c$tlsfp=TLSFPStorage();
    if ( is_orig == T  ) {
        for ( i in curves ) {
            if ( curves[i] in grease ) {
            next;
            }
            if ( c$tlsfp$e_curves == "" ) {
                c$tlsfp$e_curves += cat(curves[i]);
            }
            else {
                c$tlsfp$e_curves += string_cat(sep,cat(curves[i]));
            }
        }
    }
}

event ssl_client_hello(c: connection, version: count, possible_ts: time, client_random: string, session_id: string, ciphers: index_v
{
    if ( !c?$tlsfp )
    c$tlsfp=TLSFPStorage();
    c$tlsfp$client_version = version;
    for ( i in ciphers ) {
        if ( ciphers[i] in grease ) {
            next;
        }
        if ( c$tlsfp$client_ciphers == "" ) {
            c$tlsfp$client_ciphers += cat(ciphers[i]);
        }
        else {
            c$tlsfp$client_ciphers += string_cat(sep,cat(ciphers[i]));
        }
    }
    local sep2 = ",";
    local ja3_string = string_cat(cat(c$tlsfp$client_version),sep2,c$tlsfp$client_ciphers,sep2,c$tlsfp$extensions,sep2,c$tlsfp$e_cur
    local tlsfp_1 = md5_hash(ja3_string);
    c$ssl$ja3 = tlsfp_1;

# LOG FIELD VALUES ##
#c$ssl$ja3_version = cat(c$tlsfp$client_version);
#c$ssl$ja3_ciphers = c$tlsfp$client_ciphers;
#c$ssl$ja3_extensions = c$tlsfp$extensions;
#c$ssl$ja3_ec = c$tlsfp$e_curves;
#c$ssl$ja3_ec_fmt = c$tlsfp$ec_point_fmt;
#
# FOR DEBUGGING ##
#print "JA3: "+tlsfp_1+" Fingerprint String: "+ja3_string;

}
```

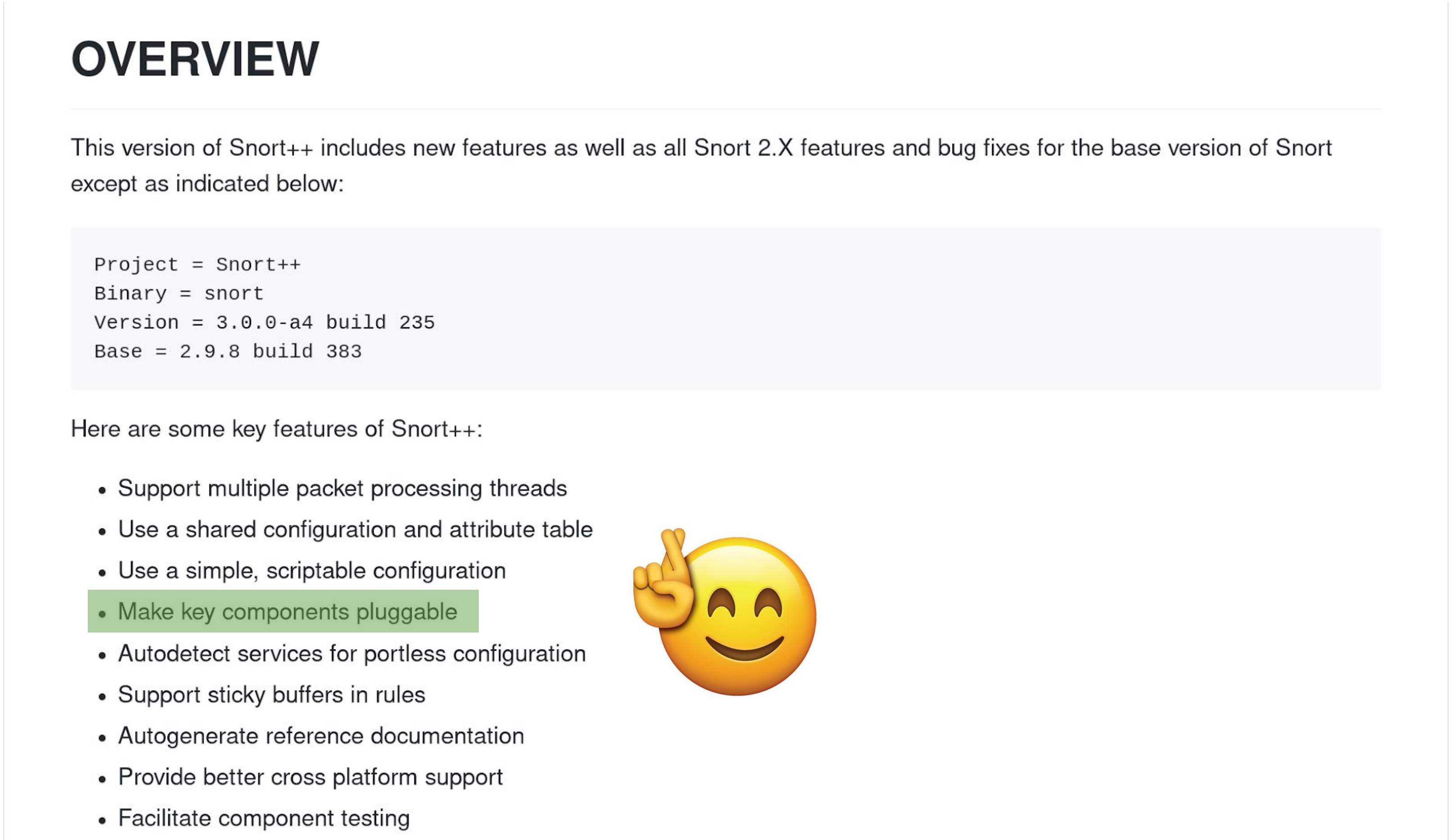
© 2018 GitHub, Inc. Terms Privacy Security Status Help — Contact GitHub Pricing API Training Blog About

Extension example: JA3 in Suricata





From https://github.com/snort3/snort3/blob/master/README.md …



OVERVIEW

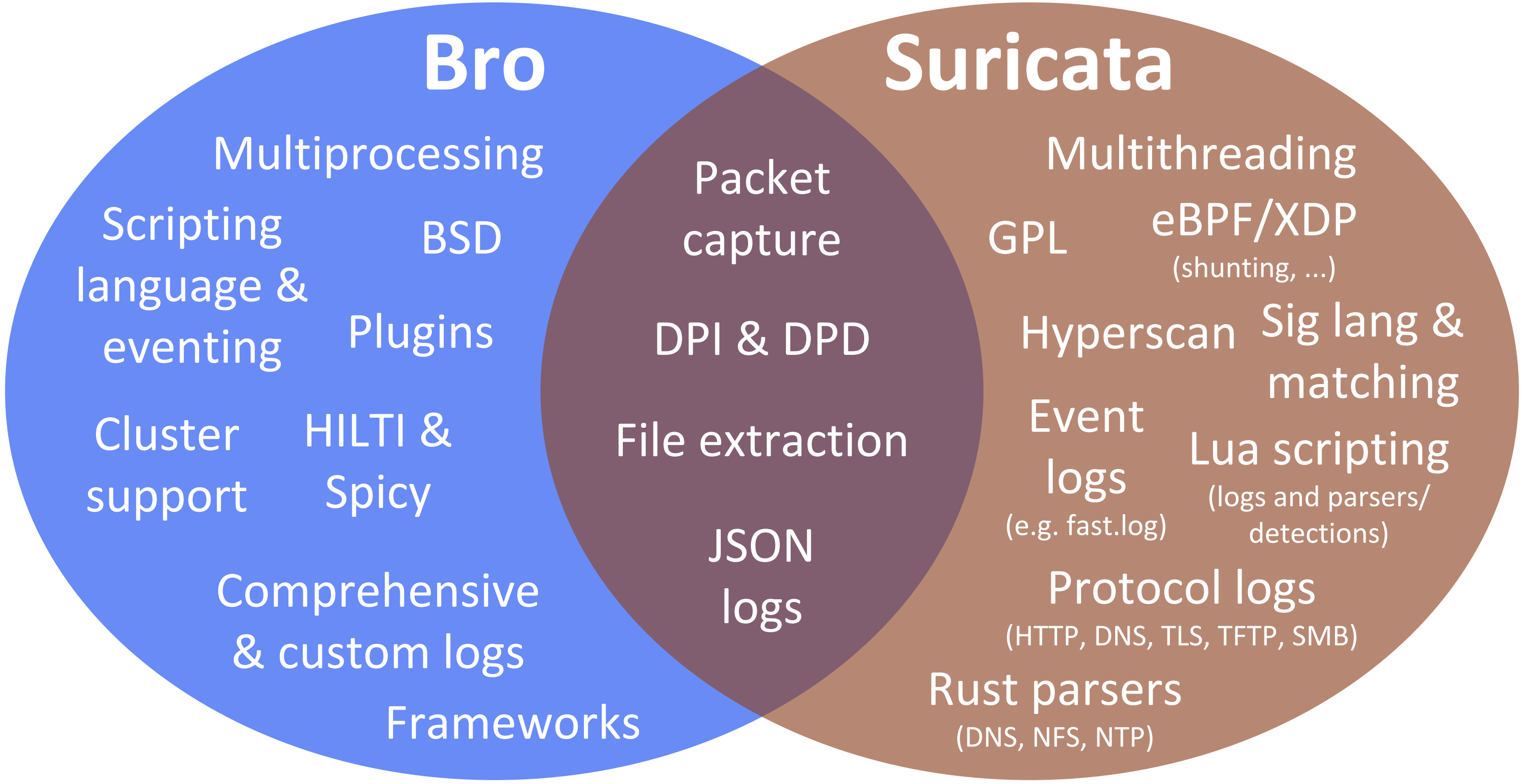
This version of Snort++ includes new features as well as all Snort 2.X features and bug fixes for the base version of Snort except as indicated below:

```
Project = Snort++
Binary = snort
Version = 3.0.0-a4 build 235
Base = 2.9.8 build 383
```

Here are some key features of Snort++:

- Support multiple packet processing threads
- Use a shared configuration and attribute table
- Use a simple, scriptable configuration
- Make key components pluggable
- Autodetect services for portless configuration
- Support sticky buffers in rules
- Autogenerate reference documentation
- Provide better cross platform support
- Facilitate component testing

Feature comparison



Bro
Suricata
Multiprocessing
Scripting language & eventing
BSD
Plugins
Cluster support
HILTI & Spicy
Comprehensive & custom logs
Frameworks
Packet capture
DPI & DPD
File extraction
JSON logs
Multithreading
GPL
eBPF/XDP (shunting, ...)
Hyperscan
Sig lang & matching
Event logs (e.g. fast.log)
Lua scripting (logs and parsers/ detections)
Protocol logs (HTTP, DNS, TLS, TFTP, SMB)
Rust parsers (DNS, NFS, NTP)

Part 3

Philosophies

Bro is a network monitor

- Nothing intrinsically implies badness
- Perfectly useful *without detecting anything*
- Just happens to be great for deeply behavioral, stateful detections

Suricata is a misuse detector

- Purpose is to detect patterns, usually of badness
- Everything in architecture ties to signature rules
- Has gained Bro-esque features because the community finds these useful, not because they’re natural for the architecture

Culturally different projects

- BSD vs GPL
- Scripting vs “rulethink”
- Research project vs off-the-shelf
- Extensibility vs all-in-one
- Roadmap prioritization

Thanks!