Snort Plug-in Development: Teaching an Old Pig New Tricks

Ben Feinstein, CISSP GCFA
SecureWorks Counter Threat Unit™

DEFCON 16
August 8, 2008
What’s In This Talk?

• Snort v2 Architecture & Internals

• Snort Plug-in Development
  ▪ Dynamic Rules
  ▪ Dynamic Preprocessors

• Snort Plug-in API
  ▪ Examples, Pitfalls, Tips

• Releasing two Dynamic Preprocessors
  ▪ ActiveX Virtual Killbits (DEMO)
  ▪ Debian OpenSSL Predictable PRNG Detection (DEMO)
Snort Basics

- Open-source IDS created by Marty Roesch
- First released for *NIX platforms 1998
- Commercialized by Sourcefire, Inc.

- Snort Inline mode now available for IPS
  - Linux Bridge + Netfilter
  - Linux ip_queue and nf_queue interfaces

- Snort v3 now making its way through Beta
  - NOT discussing plug-ins for v3
  - NOT discussing v3 architecture (ask Marty)
Snort v2 Architecture

The Basics

• Highly modularized for extensibility

• Snort Rules & The Rules Matching Engine
  ▪ SF Engine Dynamic Plug-in
  ▪ Detection Plug-ins – implement/extend rules language

• Output Plugins
  ▪ Unified / Unified2
  ▪ Syslog
  ▪ Others

• Preprocessors
  ▪ Detection (i.e. alerting)
  ▪ Normalization (i.e. decoding)
Snort v2 Architecture
Run-time (Dynamic) Extensions

- Dynamic Preprocessors
  - Define a packet processing callback
  - Preprocessor local storage
  - Stream-local storage

- Dynamic Rules
  - Writing Snort rules in C
  - v2.6.x (?), added ability to register a C callback
    - Before, only useful as form of rule obfuscation
  - Used by some commercial Snort rulesets
  - Relatively straight forward to RE using IDA Pro
Other Snort Internals of Interest
Unified2 Output Formats

• Alert vs. Log
  ▪ Log contains packet capture data in addition

• Unified2 is extensible
  ▪ Additional data in simple Length|Value encoding

• Does your detection preprocessor need to log additional alert data?
  ▪ Use Unified2!

• Examples
  ▪ Portscan Alerts
  ▪ Preprocessor Stats
Snort Plug-in Development

Getting Started

• Familiarity with the C language

• Lack of code-level documentation
  ▪ What is available is out of date

• Snort-Devel mailing list
  ▪ Sourcefire developers are very responsive, thanks!
  ▪ Do your homework before mailing the list.
  ▪ You will get a better response and save everybody time.

• Source contains very basic examples
  ▪ Dynamic Rules
  ▪ Dynamic Preprocessor
Snort Plug-in Development
Getting Started, Continued

• Use the Source!

• Examine existing plug-ins
  ▪ SMTP
  ▪ DNS
  ▪ SSH
  ▪ SSL
  ▪ HTTP Inspect (bigger)

• Write small blocks of code and (unit) test them

• Ask questions on the Snort-Devel mailing list
Snort Development Environment

- Snort 2.8.x source tarball

- CentOS 5
  - gcc 4.1
  - glibc 2.5

- GNU Autoconf 2.61
  - CentOS 5 packages older version 2.59

- GNU Automake 1.10
  - CentOS 5 packages older version 1.9.6
Snort Dynamic Rules

Background

• Key header file "sf_snort_plugin_api.h"
  ▪ Defines C-struct equivalents to rule syntax

• You define global variable
  ▪ Rules *rules[]
  ▪ Framework will handle the rest

• Makefile
  ▪ Compile C files into object code
  ▪ Use GNU Libtool to make dynamic shared objects

• Dynamically loaded by Snort at run-time
Snort Dynamic Rules

Configuration

- Snort config
  - `--dynamic-detection-lib <.so file>`
  - `--dynamic-detection-lib-dir <path to .so file(s)>`

- Snort can create stub rules files for all loaded dynamic rules
  - `--dump-dynamic-rules <output path>`

- "meta rules" must be loaded in Snort rules file
  - `alert tcp any any -> any any (msg:"Hello World!"; [...] metadata : engine shared, soid 3|2000001; sid:2000001; gid:3; rev:1; [...] )`
Snort Plug-in API

- Different C structs for each rule option in rules language
- A Rule Option is a Union of different specific rule opt structs
- Rule struct w/ NULL-terminated array of Rule Options
  - Rule Header
  - Rule References
- Functions for matching
  - content, flow, flowbits, pcre, byte_test, byte_jump
- Function to register and dump rules
static ContentInfo sid109content =
{
    (u_int8_t *)"NetBus", /* pattern to search for */
    0, /* depth */
    0, /* offset */
    CONTENT_BUF_NORMALIZED, /* flags */
    NULL, /* holder for aho-corasick info */
    NULL, /* holder for byte representation of "NetBus" */
    0, /* holder for length of byte representation */
    0 /* holder of increment length */
};
static RuleOption sid109option2 =
{
    OPTION_TYPE_CONTENT,
    {
        &sid109content
    }
};

ENGINE_LINKAGE int contentMatch(void *p, ContentInfo* content, const u_int8_t **cursor);
static PCREInfo activeXPCRE =
{
    "<object|\snew\s+ActiveX(Object|Control)",
    NULL,
    NULL,
    PCRE_CASELESS,
    CONTENT_BUF_NORMALIZED
};
static RuleOption activeXPCREOption =
{
    OPTION_TYPE_PCRE,
    {
        &activeXPCRE
    }
};
Snort Plug-in API
PCRE Matching (Continued)

ENGINE_LINKAGE int pcreMatch(void *p, PCREInfo* pcre, const u_int8_t **cursor);
static FlowFlags activeXFlowFlags = {
    FLOW_ESTABLISHED|FLOW_TO_CLIENT
};

static RuleOption activeXFlowOption = {
    OPTION_TYPE_FLOWFLAGS,
    {
        &activeXFlowFlags
    }
};

ENGINE_LINKAGE int checkFlow(void *p, FlowFlags *flowFlags);
extern Rule sid109;
extern Rule sid637;

Rule *rules[] =
{
    &sid109,
    &sid637,
    NULL
};

/* automatically handled by the dynamic rule framework */
ENGINE_LINKAGE int RegisterRules(Rule **rules);
Optional C packet processing callback
  - Returns RULE_MATCH or RULE_NOMATCH

sf_snort_plugin_api.h:

typedef int (*ruleEvalFunc)(void *);

typedef struct _Rule {
  [...]
  ruleEvalFunc evalFunc;
  [...]
} Rule;
my_dynamic_rule.c:

#include "sf_snort_plugin_api.h"
#include "sf_snort_packet.h"

int myRuleDetectionFunc(void *p);

Rule myRule = {
    [...] ,
    &myRuleDetectionFunc ,
    [...] 
};
my_dynamic_rule.c (con't):

```c
int myRuleDetectionFunc(void *p) {
    SFSnortPacket *sp = (SFSnortPacket *) p;

    if ((sp) && (sp->ip4_header.identifier % (u_int16_t)2))
        return RULE_MATCH;

    return RULE_NOMATCH;
}
```

- Question for Audience: What does this do?
Snort Dynamic Preprocessors

Background

- Another key header file: "sf_dynamic_preprocessor.h"

- Key struct: "DynamicPreprocessorData"
  - Typically defined as extern variable named "_dpd"

- Contains:
  - Functions to add callbacks for Init / Exit / Restart
  - Internal logging functions
  - Stream API
  - Search API
  - Alert functions
  - Snort Inline (IPS) functions
void SetupActiveX(void) {
    _dpd.registerPreproc("activex", ActiveXInit);
}

static void ActiveXInit(char *args) {
    _dpd.addPreproc(ProcessActiveX,
                PRIORITY_TRANSPORT, PP_ACTIVEX);
}

static void ProcessActiveX(void* pkt, void* contextp) {
    [...] 
    _dpd.alertAdd(GENERATOR_SPP_ACTIVEX,
                ACTIVEX_EVENT_KILLBIT, 1, 0, 3,
                ACTIVEX_EVENT_KILLBIT_STR, 0);
    return;
}
Snort Plug-in API
Using Rules Within a Dynamic Preprocessor

• We can try calling rule option matching functions directly, but need internal structures first properly initialized.

• Use dummy Rule struct and ruleMatch():
  ▪ ENGINE_LINKAGE int ruleMatch(void *p, Rule *rule);

• RegisterOneRule(&rule, DONT_REGISTER_RULE);

• Confusing, huh?
• RegisterOneRule will setup Aho-Corasick and internal ptrs
• But we don't always want to register the rules as an OTN

• So, pass in DONT_REGISTER_RULE. See?
SecureWorks Snort Plug-ins

- Available under:

- Released under GPLv2 (or later)

- No Support

- No Warranty

- Use at Your Own Risk

- Feedback is appreciated!
ActiveX Detection Dynamic Preprocessor

- Inspects web traffic for scripting instantiating "vulnerable" ActiveX controls
  - As based on public vulnerability disclosures

- Preprocessor configuration points to local DB of ActiveX controls
  - Listed by CLSID and optionally method/property
  - XML format (I know, I know...)

- Looks at traffic being returned from HTTP servers
  - ActiveX instantiation and Class ID
  - Access to ActiveX control's methods / properties
ActiveX Detection Dynamic Preprocessor
Continued

- Can presently be bypassed
  - JavaScript obfuscation
  - HTTP encodings
  - But many attackers still using plain CLSID!

- Future Snort Inline support
  - Drop or TCP RST the HTTP response

- Leveraging of normalization done by HTTP Inspect

- Enhance to use Unified2 extra data to log detected domain name
ActiveX Detection Dynamic Preprocessor

Internals

• Uses matchRule(Rule*) from Snort Plug-in API
  ▪ Very convenient
  ▪ Not the most efficient

• Performs naïve linear search of CLSIDs
  ▪ Enhance to reuse HTTP Inspect's high-performance data-structures?

• Uses Snort's flow match

• Performs content matching and PCRE matching
ActiveX Detection Dynamic Preprocessor

Live Demo
Debian OpenSSL Predictable PRNG Vuln
CVE-2008-0166

- Lack of sufficient entropy in PRNG delivered by Debian's OpenSSL package

- Go see Luciano Bello and Maximiliano Bertacchini's talk!
  - Saturday, 13:00 – 13:50, Track 4

- One of the coolest vulns of 2008!
  - Pwnie for Mass Ownage!

- Keys generated since 2006-09-17

- Keys generated with Debian Etch, Lenny or Sid
  - Downstream distros such as Ubuntu also vulnerable
Debian OpenSSL Predictable PRNG Vuln

Dilbert (source: H D Moore, metasploit.com)
Debian OpenSSL Predictable PRNG Vuln

XKCD (source: H D Moore, metasploit.com)

```c
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
    // guaranteed to be random.
}
```
Debian OpenSSL Predictable PRNG Vuln
It’s Bad!

• From the Debian Wiki (http://wiki.debian.org/SSLkeys):

  "... any DSA key must be considered compromised if it has been used on a machine with a ‘bad’ OpenSSL. Simply using a ‘strong’ DSA key (i.e., generated with a ‘good’ OpenSSL) to make a connection from such a machine may have compromised it. This is due to an ‘attack’ on DSA that allows the secret key to be found if the nonce used in the signature is known or reused."

• H D Moore was all over this one with a quickness!
  - Metasploit hosting lists of brute-forced 'weak' keys
Debian OpenSSL Predictable PRNG Vuln
Detection & Mitigation

- You scanned your assets for SSH / SSL servers using the blacklisted keys, right? (Tenable Nessus)

- You scanned all user home dirs for blacklisted SSH keys?
  - Debian ssh-vulnkey tool

- You scanned all user homedirs, Windows Protected Storage, and browser profiles for blacklisted SSL certs, right?

- But what about connections to external servers that use the vulnerable Debian OpenSSL?
Debian OpenSSL Predictable PRNG Preproc.

• Goal: Detect SSH Diffie-Hellman Key Exchange (KEX) where client and/or server are OpenSSH linked against vulnerable Debian OpenSSL

• Just that detective capability is valuable
  • Even w/ great technical controls in place, you're likely missing:
    • Users connecting to external servers using bad OpenSSL
    • Connections to/from external hosts that use bad OpenSSL

• What else can we do?
Debian OpenSSL Predictable PRNG Preproc. Continued

- Goal: Have preprocessor(s) "normalize" traffic by brute-forcing the DH key exchange, decoding both sides of session on-the-fly.
  - Snort rule matching engine and other preprocessors can then inspect unencrypted session
  - Unencrypted sessions can be logged (Unified or PCAP)

- Potential issue w/ source code release
  - Controls on the export of cryptanalytic software (US)
Debian OpenSSL Predictable PRNG Preproc.

Credits

- Alexander Klink
  - http://seclists.org/fulldisclosure/2008/May/0592.html
  - http://www.cynops.de/download/check_weak_dh_ssh.pl.bz2

- Paolo Abeni, Luciano Bello & Maximiliano Bertacchini
  - Wireshark patch to break PFS in SSL/TLS
  - https://bugs.wireshark.org/bugzilla/show_bug.cgi?id=2725

- Raphaël Rigo & Yoann Guillot
  - New work on SSH and Debian OpenSSL PRNG Vuln
  - Unknown to me until hearing about it at DEFCON
  - http://www.cr0.org/progs/sshfun/
Diffie-Hellman Key Exchange for SSH
Do the Math!

- A way for two parties to agree on a random shared secret over an insecure channel.
- Server sends to Client
  - $p$ – large prime number
  - $g$ – generator of the field $(\mathbb{Z}_p)^*$ (typically 0x02)
- Client generates random number $a$
  - Calculates $g^a \mod p$
  - Sends calculated value to server
- Server generates random number $b$
  - Calculates $g^b \mod p$
  - Sends calculated value to client
Diffie-Hellman Key Exchange for SSH

Do the Math! (2)

- DH shared secret is defined as both a function of $a$ and of $b$, so only parties that know $a$ or $b$ can calculate it.

- Client
  - knows $g$, $a$ and $g^b \mod p$
  - Calculates shared secret as $(g^b)^a = g^{ab} \mod p$

- Server
  - knows $g$, $b$ and $g^a \mod p$
  - Calculates shared secret as $(g^a)^b = g^{ab} \mod p$
Diffie-Hellman Key Exchange for SSH
Do the Math! (3)

- Eavesdropper knows $g, g^a \mod p$ and $g^b \mod p$
- Can't calculate $g^{ab} \mod p$ from $g^a \mod p$ and $g^b \mod p$
- Must solve the discrete logarithm problem
  - No known (non-quantum) algorithm to solve in polynomial time

  *Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*
  - Peter W. Shor, AT&T Research
Diffie-Hellman Key Exchange for SSH
Do the Math! (4)

- Encryption IVs and Keys generated from DH shared secret
- $V_C, V_S$ – Client / Server's SSH version announce string
- $I_C, I_S$ – Client / Server's SSH_MSG_KEXINIT message
- $K_S$ – Server's Public Host Key

- $H = \text{hash}(V_C || V_S || I_C || I_S || K_S || g^a \mod p || g^b \mod p || g^{ab} \mod p)$
- SSH session_id = H of initial DH key exchange

- IV client to server: hash($g^{ab} \mod p || H || "A" || \text{session}_id$)
- IV server to client: hash($g^{ab} \mod p || H || "B" || \text{session}_id$)
- Enc Key client to server: hash($g^{ab} \mod p || H || "C" || \text{session}_id$)
- Enc Key server to client: hash($g^{ab} \mod p || H || "D" || \text{session}_id$)
The Debian OpenSSL PRNG and SSH DH GEX

- If OpenSSH client or server is linked against vulnerable Debian OpenSSL
  - $a$ or $b$ is completely predictable based on ProcessID of OpenSSH

- We can quickly brute force $a$ or $b$.
  - Only 32768 possibilities!

- If we know $a$ or $b$, we can calculate DH shared secret
  - $g^{ab} \mod p = (g^b)^a = (g^a)^b$

- Once we know the DH shared secret, we have everything needed to decrypt the SSH session layer!
The Debian OpenSSL PRNG and SSH DH GEX

The Impact

- Tunneled Clear Text Passwords are compromised
  - ...if *either* client or server is using vulnerable OpenSSL
  - RSA / DSA public key authentication is not affected

- Files or other data protected by SSH Session layer are compromised
  - ...if *either* client or server is using vulnerable OpenSSL

- Observers can easily tell if either client or server is using vulnerable OpenSSL
  - ...and proceed to decrypt the stream
Detection of SSH Diffie-Hellman KEX using vulnerable Debian OpenSSL

Live Demo
Snort Futures

• Snort v3
  ▪ Complete redesign from the ground up
  ▪ Extremenly flexible and extensible architecture
  ▪ Snort 2.8.x matching engine plugs in as module
  ▪ HW optimized packet acquisition can be plugged in
  ▪ Lua programming language!

• Snort 2.8.3 (Release Candidate)
  ▪ Enhancements to HTTP Inspect
    • Normalized Buffers for Method, URI, Headers, Cookies, Body
    • Content and PCRE matching against new buffers
  ▪ New HTTP normalization exposed in Snort Plug-in API
Wrapping It All Up

- Snort is a powerful framework to work with
  - APIs for alerting, logging, Streams, matching
  - Why reinvent the wheel?

- Hopefully, you can take away needed info to start writing your own plug-ins.

- Read the source code of other plug-ins, ask questions.

- Snort v2 is still evolving. If the APIs don't support something you (and potentially others?) really need, ask and ye may receive.
Thanks to DT, the Goons and everyone who made DEFCON a reality this year!
Greetz to DC404, Atlanta's DC Group!
Speakers: dr.kaos, Carric, David Maynor, Scott Moulton
& Adam Bregenzer
And our very own Goon, dc0de!
Questions?

bfeinstein@secureworks.com