```
|------------------------------------------------------------------------------|
|-------------------- Fully arbitrary 802.3 packet injection ------------------|
|-------------------- Maximizing the Ethernet attack surface ------------------|
|------------------------------------------------------------------------------|
|------------------------------------------------------------------------------|
|----------------------            Inverse Path            --------------------|
|----------------------                                    --------------------|
|----------------------   Andrea Barisani                  --------------------|
|----------------------   <lcars_at_inversepath_dot_com>   --------------------|
|----------------------                                    --------------------|
|----------------------   Daniele Bianco                   --------------------|
|----------------------   <danbia_at_inversepath_dot_com>  --------------------|
|----------------------                                    --------------------|
|----------------------  http://dev.inversepath.com/802.3  --------------------|
|--------------------------------------------------------------------20130712--|
```

--[Contents

1.   Introduction
2.   Scope
3.   IEEE 802.3 frame characteristics
4.   Challenges in arbitrary packet sniffing and injection
5.   Motivation
6.   Hardware setup
7.   Passive network tap evasion through SFD manipulation
8.   SFD parsing anomalies
9.   Packet-In-Packet on wired Ethernet
10. Future work

I.   References & Notes
II. Code
III. Links

--[1. Introduction

It is generally assumed that sending and sniffing arbitrary Fast Ethernet packets can be performed with standard Network Interface Cards (NIC) and generally available packet injection software. However, full control of frame values such as the Frame Check Sequence (FCS) or Start-of-Frame delimiter (SFD) has historically required the use of dedicated and costly hardware.

This paper dissects Fast Ethernet layer 1 & 2 presenting novel attack techniques supported by an affordable hardware setup that, using customized firmware, allows fully arbitrary frame injection.

This research expands the ability to test and analyse the full attack surface of networked embedded systems, with particular attention on automation, automotive and avionics industries. Application of such attacks is particularly appealing against NICs with hard and soft Media Access Control (MAC) on industrial embedded systems.

Additionally the paper illustrates how specific frame manipulations and attack conditions can trigger SFD parsing anomalies, Ethernet Packet-In-Packet injection and passive network tap evasion.

--[2. Scope

The paper assumes Fast Ethernet standard 100BASE-TX with full duplex mode of operation, Auto-Negotiation is ignored.

--[3. IEEE 802.3 frame characteristics

In telecommunications the common necessity of transmitting control information is generally handled with out-of-band signalling and/or in-band signalling. The Ethernet protocol employs control signals that can be seen as out-of-band from the perspective of certain layers.

Following the OSI model, the IEEE 802.3 specification defines a physical (PHY) layer and a data link layer, named Media Access Control (MAC). This separation is physically followed in NIC implementations which employ two separate integrated circuits handling the respective layers, a PHY chip and a MAC chip, generally connected to each other using the Media Independent Interface (MII) standard.

The PHY layer is responsible for handling all media dependent sub-layers such as the Physical Coding Sublayer (PCS), which concerns the encoding and synchronization of symbol streams, detecting the Start-of-Stream Delimiter (SSD) and the End-of-Stream Delimiter (ESD) or handling the optional Auto-Negotiation protocol.

The MAC layer responsibilities include discarding malformed frames, appending or removing the Preamble and Start-of-Frame Delimiter (SFD), and checking the Frame Check Sequence (FCS).

Due to this separation, different layers see the IEEE 802.3 Ethernet Frame in different ways, the following diagram illustrates this separation.

```
       <- Physical Layer Stream ------------------------------------->
 --------------------------------------------------------------------------
| Idle | SSD | Preamble | SFD | DA | SA | ET | MAC Data | FCS | ESD | Idle |
 --------------------------------------------------------------------------
             <- MAC Packet ------------------------------------->
                              <- MAC Frame ------------------->
```

The physical stream contents are encoded using the Multi-Level Transition 3 scheme (MLT3), this encoding prevents clock recovery issues by always containing at least two '1's, resulting in at least two MLT3 waveform transitions.

Additionally the separation between the PHY and MAC layers requires the transmission of PHY signalling data to be unambiguously encapsulated in relation to the MAC Frame. This is necessary to prevent data within the MAC Frame to collide with PHY handled symbols such as the ESD. The separation between PHY signalling codes and frame data is accomplished by encoding 4 bits of data with 5 bits (4B/5B encoding), allowing the transmission of 32 unique symbols (2^5) of which 22 are actually used (16 for the MAC layer and 6 additional symbols reserved for the PHY layer). For this reason the 100 Mb/s actual bit rate is 125 Mb/s.

The available symbols, and frame elements earlier depicted, are described in the following tables.

```
--------------------------------------------------------------------
0 11110    4 01010    8 10010    C 11010    I 11111 (Idle)
1 01001    5 01011    9 10011    D 11011    J 11000 (SSD, Part 1)
2 10100    6 01110    A 10110    E 11100    K 10001 (SSD, Part 2)
3 10101    7 01111    B 10111    F 11101    T 01101 (ESD, Part 1)
                                            R 00111 (ESD, Part 2)
                                            H 00100 (Transmit Error)
--------------------------------------------------------------------
```

Definition	Values	5B encoding	Description
Idle	/I/	11111	inter frame gap
SSD	/J/K/	11000 10001	PHY frame beginning marker
Preamble	/5/5/	01011 01011	MAC Preamble (6 octets of 0x55)
SFD	/D/5/	11011 01011	MAC frame beginning marker (0xd5)
DA / SA	0-F		MAC Destination / Source Address
ET	0-F		EtherType or Length
MAC Data	0-F		Data
FCS	0-F		Frame checksum
ESD	/T/R/	01101 00111	PHY frame end marker

We can see that, when transmitting or receiving a packet, it is impossible for the MAC Frame to include symbols that match PHY signalling codes, however MAC signalling codes (Preamble, SFD) are represented with the same symbols allowed within MAC Frame.

The MAC Preamble value is shown to have 6 octets because, on transmission, the first 8 bits are replaced with the SSD. This replacement is reversed on reception, therefore technically the MAC Preamble is composed of 7 octets.

The actual frames sent on the wire are scrambled to minimize cross-talk, therefore the levels monitored on the wire with an oscilloscope cannot be directly translated with the shown 4B/5B table but require descrambling first.

--[4. Challenges in arbitrary packet sniffing and injection

The layer separation between PHY and MAC poses certain challenges to a completely arbitrary packet transmission and reception.

The Idle, SSD and ESD components of the Ethernet Frame are generally not available at the MAC layer, in either direction. Similarly the Preamble and SFD are generally not available at the OS driver layer or even by modifications of the user loadable NIC firmware.

The FCS is generally not included in packets handled by the OS as its check, or computation, is offloaded to the MAC. Additionally packets with an invalid FCS are discarded by the MAC and never sent to the OS.

The FCS reception behaviour can be easily changed by either patching the driver or, in certain cases, taking advantage of specific driver flags that might be supported. As an example the Linux e1000e driver for Intel PRO/1000 PCI-Express Gigabit Ethernet supports the disabling of specific offloading features, allowing to receive the FCS value as well as packets with invalid values (rx-fcs and rx-all offload parameters which can be set with ethtool).

On the other hand, still for the specific e1000e driver, the injection of packets with a malformed FCS requires a custom driver patch (see Code appendix).

The arbitrary manipulation of Preamble and SFD values can only be accomplished in environments where full control of the MAC layer is possible, therefore a normal computer, even with driver patches, is not sufficient.

--[5. Motivation

The difficulties in sending Ethernet packets with arbitrary Preamble and SFD naturally inspires a challenge in evaluating that expectations concerning these packet values can be leveraged to trigger unexpected behaviours with security implications.

Devices that implement software MACs, such as FPGA-based dedicated embedded systems often encountered in automation, automotive and avionics industries, or dedicated Ethernet multiplexers are appealing targets.

In fact, the motivation for publishing this work results from specific needs raised, and verified, during the security testing of such class of devices by the authors.

--[6. Hardware setup

A dedicated hardware setup, using an FPGA or a sufficiently powerful microcontroller, is necessary to implement a software MAC with the programmable functionality required to craft fully arbitrary IEEE 802.3 Ethernet frames at the MAC layer.

The authors have identified in the XMOS XC-2 Ethernet Kit an affordable and easy off-the-shelf solution that, with customized firmware, well serves the task of sending arbitrary frames. The board implements a single four-core programmable XMOS processor attached to a SMSC LAN8700C-AEZG Ethernet Transceiver, which acts as the PHY.

The provided development resources include firmware code that can be patched to send raw MAC Packets, including Preamble, SFD and FCS. Please see the Code appendix to download the source code of the custom packet injector developed for this project.

The injector, once executed, prompts for the raw MAC Packet payload, starting from Preamble and including the final FCS, and the packet count.

```
$ xrun --id 0 --io ethernet.xe
injector start
Enter payload
55555555555555d5001f1637f2ff00000000000108004500003900004000400616bb0
a0108020a010801029a029a0000000000000000500200004f55000000000000000000
000000666f6f62617200271232ab
Enter repeat count (0 = unlimited)
0
Sending payload unlimited times (83 bytes)
```

--[7. Passive network tap evasion through SFD manipulation

The use of passive network taps [1] to monitor Ethernet communications, with an Intrusion Detection System for instance, is a fairly common practice when active tap ports are not available or in certain operating environments that try to minimize the complexity and intrusiveness of the tap.

The ability to manipulate the Preamble and the SFD allows us to fingerprint specific MAC implementations and explore interpretation ambiguities that can be leveraged to inject Ethernet frames valid for the first destination but otherwise discarded by the monitoring NIC.

The following table illustrates Preamble & SFD handling behaviours fingerprinted for a small set of devices, the shown values represent the only accepted Preamble & SFD combinations.

Device	MAC	Preamble	SFD
Intel DH61DL	82579V GE	any^ + 0x55 * N	0xd5
Intel x200s	82567LM	any^ + 0x55 * N	0xd5
Linksys WRT54GCV3	BCM5354	any^ + 0x55 * N	0xd5
Planex	RTL8309SB	any^ * 1-N	0xd5
Netgear JFS524E	Unknown	any^ * 1-N	0xd5

Cisco Catalyst 2950	Intel HBLXT9785	any^ * 1-N	0xd*
TP-Link TL-WR941ND	Marvell 88E6060	any^ * 1-N	0xd5~

^ any value without the least significant nibble set to 0b1101 (0xd), which triggers a misaligned packet

~ not quite, see the next section for details

It is evident that the compliant Preamble length is never enforced, at least one octet is necessary to allow conversion of the first 8 bits to SSD.

As an example, in a scenario where the NIC attached to the tap is based on the 82579V GE MAC, commonly found on Intel motherboards, while the rest of the connection is routed via a Cisco Catalyst 2950 switch, detection can be evaded by sending packets with a SFD value set to 0xd4. The Catalyst switch would happily accept the packet while the Intel NIC would silently discard it without any visibility to the OS.

--[8. SFD parsing anomalies

Certain network devices exhibit parsing anomalies caused by specific expectations in the Finite-state machine (FSM) of their MAC implementation. Such anomalies can be leveraged to cause Denial of Service conditions (DoS) or more peculiar behaviours.

As an example a curious race condition can be triggered on Marvell 88E6060 based network switches. Let us consider the following special packet, here represented with offset relative to the beginning of the MAC Frame.

```
-0002  55 d4                                             | Preamble + !SFD
 0000  00 1f 16 37 b1 3d 00 00 a0 ea 8e 91               | dst + src MAC addrs
 000c  08 00                                             | EtherType (IPv4)
 000e  00 55 d5                                          | Preamble + SFD
 0011  00 1f 16 37 b1 3d 00 00 00 00 00 01               | dst + src MAC addrs
 001d  08 00                                             | Ethertype (IPv4)
 001f  45 00 00 39 00 00 40 00 40 06 16 bb 0a 01 08 02 | IPv4 payload
 002f  0a 01 08 01 02 9a 02 9a 00 00 00 00 00 00 00 00 | IPv4 payload
 003f  50 02 00 00 4f 55 00 00 00 00 00 00 00 00 00 00 | IPv4 payload
 004f  00 00 72 61 63 65 6d 65 00                        | IPv4 payload
 0058  4a 84 9e 8d                                       | FCS
```

The packet has peculiar characteristics:
- the beginning of the frame does not have valid SFD (0xd4)
- a shortened Preamble (at 0x0f) and valid SFD (at 0x10) are later included
- both 0x00-0x0d and 0x11-0x1e ranges include a valid Ethernet frame header
- a CRC32 collision is generated so that the same FCS is valid for the two frames

To summarize two frames can be seen at the following offsets:
- frame 1: 0x00-0x0d (Ethernet frame header) + 0x0e-0x57 (Payload) + FCS
- frame 2: 0x11-0x1e (Ethernet frame header) + 0x1f-0x57 (Payload) + FCS

The CRC32 collision is calculated using the excellent CRC32 compensation tool developed by Julien Tinnes [2] and a custom helper script (see Code appendix).

The incorrect SFD, when sent multiple times to a Marvell 88E6060 based device, triggers a race condition that causes the alignment of the valid packet to be randomly detected. As both alignments exhibit a valid frame and FCS, the two cases are transmitted to the OS.

The same packet, sent 10 times, is received as follows:

```
11:56:50 IP0 bad-hlen 0
11:56:51 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
11:56:52 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
11:56:53 IP0 bad-hlen 0
11:56:54 IP0 bad-hlen 0
11:56:55 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
11:56:56 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
11:56:57 IP0 bad-hlen 0
11:56:58 IP0 bad-hlen 0
11:56:59 IP0 bad-hlen 0
```

The anomaly is evident when the same transmission pattern is repeated and
received as follows:

```
11:57:09 IP0 bad-hlen 0
11:57:10 IP0 bad-hlen 0
11:57:11 IP0 bad-hlen 0
11:57:12 IP0 bad-hlen 0
11:57:13 IP0 bad-hlen 0
11:57:14 IP0 bad-hlen 0
11:57:15 IP0 bad-hlen 0
11:57:15 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
11:57:16 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
11:57:17 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
```

The following verbose output shows the packet payload, note that the FCS is
included in the tcpdump output due to the rx-fcs being enabled via ethtool.

```
11:56:57.300491 00:00:a0:ea:8e:91 > 00:1f:16:37:b1:3d, ethertype Ipv4 (0x0800),
length 92: IP0 bad-hlen 0
        0x0000:  001f 1637 b13d 0000 a0ea 8e91 0800 0055
        0x0010:  d500 1f16 37b1 3d00 0000 0000 0108 0045
        0x0020:  0000 3900 0040 0040 0616 bb0a 0108 020a
        0x0030:  0108 0102 9a02 9a00 0000 0000 0000 0050
        0x0040:  0200 004f 5500 0000 0000 0000 0000 0000
        0x0050:  0072 6163 656d 6500 4a84 9e8d

11:56:56.340515 00:00:00:00:00:01 > 00:1f:16:37:b1:3d, ethertype IPv4 (0x0800),
length 75: (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length
57) 10.1.8.2.666 > 10.1.8.1.666: Flags [S], cksum 0x4f55 (correct), seq 0:17,
win 0, length 17
        0x0000:  001f 1637 b13d 0000 0000 0001 0800 4500
        0x0010:  0039 0000 4000 4006 16bb 0a01 0802 0a01
        0x0020:  0801 029a 029a 0000 0000 0000 0000 5002
        0x0030:  0000 4f55 0000 0000 0000 0000 0000 0000
        0x0040:  7261 6365 6d65 004a 849e 8d
```

In the Marvell 88E6060 case the race condition seems to happen at a MAC layer,
before decision making processes concerning VLAN or port addressing, therefore
there are no security implications to the otherwise curious race condition.

This class of parsing anomalies however represents a worthy attack surface on
devices where the race condition happens to be triggered after filtering or
routing logic that is involved in security relevant decision making.

--[9. Packet-In-Packet on wired Ethernet

The injection of raw frames at Layer 1 has been successfully explored by Travis
Goodspeed et al. [3] for IEEE 802.15.4 frames, exploiting the intrinsic signal
degradation characteristics of wireless signals.

The same kind of injection would be extremely appealing for IEEE 802.3 frames though the reliability and extremely low error rate of wired cables make it unrealistic.

We explored the possible conditions that would allow successful Packet-In-Packet injection on Ethernet frames. While slim, it turns out that there is a chance of reliable Packet-In-Packet injection on Ethernet devices. The scenario requires extremely narrow conditions but is nonetheless presented for its academic, historical and entertainment value.

The connection between a PHY and MAC is generally implemented with Media Independent Interface (MII) or Reduced MII (RMII), both standards use transmit enable (TX_EN) signals to indicate, as the name suggests, that valid data is presented on TXD signals by the MAC.

When a link status change happens, due to reboot/startup of the device, link speed change or cable re-plugging, the PHY allows the transmission of a MAC Packet "in flight", as TX_EN is asserted regardless of the PHY status.

For this reason a packet, transmitted during a link state change, has the chance of having its original Preamble and SFD missed, leaving the first available values within the payload, which match Preamble and SFD parsing rules, to be treated as such.

The exploitation of this behaviour does not require any dedicated hardware as it can be accomplished with standard IP packets, even routed, transmitted towards a vulnerable device. Of course in order to succeeded the transmitter would need means to trigger the link state change, either by intermediate router/switch reboot vulnerability or other attacks...or simply wait for somebody to unplug and re-plug a cable on an intermediate switch between the router/firewall, target of the bypass, and the final target.

In order to illustrate the attack let us consider the following packet transmitted from a remote origin and routed towards a victim, the packet is a DNS request representing legitimate traffic towards the victim server.

```
17:47:15.972801 00:1f:16:37:b1:3d > 00:22:6b:dc:c6:55, ethertype IPv4 (0x0800),
length 1104: (tos 0x0, ttl 64, id 20574, offset 0, flags [none], proto UDP
(17), length 1090)
    192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
        0x0000:  0022 6bdc c655 001f 1637 b13d 0800 4500  ."k..U...7.=..E.
        0x0010:  0442 505e 0000 4011 62f1 c0a8 0001 c0a8  .BP^..@.b.......
        0x0020:  420a 927d 0035 0024 0000 c007 0100 0001  B..}.5.$........
        0x0030:  0000 0000 0000 0667 6f6f 676c 6503 636f  .......google.co
        0x0040:  6d00 0001 0001 0000 749c 9b85 0000 0000  m.......t.......
        0x0050:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        .....
        0x01f0:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        0x0200:  2165 c8fe 0000 0000 0000 0000 0000 0000  !e.............
        0x0210:  0000 0000 0000 0000 0000 0000 0000 0000  ................
        .....
        0x0400:  0055 5555 5555 5555 d500 1f16 37f2 ff00  ..UUUUUU....7...
        0x0410:  1f16 37b1 3d08 0045 0000 3900 0040 0040  ..7.=..E..9..@.@
        0x0420:  0616 bb0a 0108 020a 0108 0102 9a02 9a00  ................
        0x0430:  0000 0000 0000 0050 0200 004f 5500 0000  .......P...OU...
        0x0440:  0000 0000 0000 0000 0066 6f6f 6261 7200  .........foobar.
```

Similarly to the packet crafted for the SFD parsing anomaly, this DNS request exhibits a nested packet at offset 0x409, preceded by a valid Preamble and SFD. The nested packet is intentionally placed at a considerable distance from the beginning of the frame, to maximize the attack chances.

The 4 bytes at offset 0x200 are tweaked in order to make the resulting FCS valid for the inner packet as well as the outer packet though assumed to have a different Time To Live (TTL) value, due to hops between the attacker and the victim, and different source and destination MAC addresses.

To create the FCS collision the source and destination MAC addresses of the routed packet inconveniently have to be brute forced or known to the attacker, on IPv6 networks that employ Modified EUI-64 this is less of an issue since the MAC can be inferred from the IP address.

The FCS compensation tweak is possible as the UDP checksum can be conveniently disabled, the User Datagram Protocol standard (RFC768) teaches us that "If the computed checksum is zero, it is transmitted as all ones (the equivalent in one's complement arithmetic). An all zero transmitted checksum value means that the transmitter generated no checksum (for debugging or for higher level protocols that don't care)". While convenient this is not a necessary condition for discovering a CRC32 collision that tweaks the FCS, it just makes it easier.

In our test setup the test packet has been continuously transmitted from a laptop, using FX's file2cable [4] tool, to the default gateway and has then been routed through one additional gateway and two Ethernet switches.

Simply unplugging and replugging the RJ45 cable on an intermediate switch creates, with an average 90% success rate, the conversion of the UDP query into a TCP SYN request which is received by the victim as follows. The same effect can be obtained by rebooting one of the intermediate switches. The longer the padding between the beginning of the frame and the injected Preamble and SFD the higher are the chances of a successful injection.

The following sequence shows the victim perspective on the received stream, we can see how the UDP DNS request becomes a TCP SYN during the link status change.

```
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 10.1.8.2.666 > 10.1.8.1.666: Flags [S], seq 0:17, win 0, length 17
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
12:04:34 IP 192.168.0.1.37501 > 192.168.66.10.53: 49159+ A? google.com. (1062)
```

The following verbose output shows the TCP SYN packet payload, which matches the contents of the DNS request starting from offset 0x409.

```
12:04:34.442052 00:1f:16:37:b1:3d > 00:1f:16:37:f2:ff, ethertype IPv4 (0x0800),
length 71: (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length
57)
    10.1.8.2.666 > 10.1.8.1.666: Flags [S], cksum 0x4f55 (correct), seq 0:17,
win 0, length 17
        0x0000:  001f 1637 f2ff 001f 1637 b13d 0800 4500  ...7.....7.=..E.
        0x0010:  0039 0000 4000 4006 16bb 0a01 0802 0a01  .9..@.@.........
        0x0020:  0801 029a 029a 0000 0000 0000 0000 5002  ..............P.
        0x0030:  0000 4f55 0000 0000 0000 0000 0000 0000  ..OU............
        0x0040:  666f 6f62 6172 00                        foobar.
```

It should be noted that (rare) combinations of transmitting side with a "lazy" PHY, which transforms the first 8 bits from the MAC to SSD without checking that it is actually replacing the first 8 bits of the Preamble, and a receiving side that demands a non arbitrary Preamble (like, for instance, the 82567LM earlier included in the SFD handling fingerprint table), prevent the injection scenario. In such cases, the beginning of the transmission would require to happen exactly at the injected Preamble, which is a rare occurrence, rather than on the padding pattern.

Of course, given the fact that when Packet-In-Packet injection occurs an arbitrary Ethernet frame can be transmitted, IEEE 802.1Q VLAN tags, MPLS labels and other layer 2 (or "2.5") protocol attributes can be inserted or manipulated.

It is relevant to note that certain classes of embedded Ethernet multiplexers/demultiplexers, that combine/decode packet streams from different domain of trusts in one single stream, are severely affected by this technique as they often rely on the layer 1 structure of the combined packet for domain separation. This becomes more relevant considering that predictable MAC addresses are often employed in industrial embedded systems.

--[10. Future work

There are several possibilities for further investigation of the described techniques. The analysis of different Ethernet modes of operation, such as gigabit, or other wired networking technologies, as well as manipulation of other factors, such as exceeding frame length and timing limits, is appealing.

The Daisho [5] project represents a promising platform for arbitrary in-line monitoring and injection on different media, including gigabit Ethernet. Further development of the presented work is likely to involve our contribution to the Daisho project.

--[I. References

[1] http://greatscottgadgets.com/throwingstar/
[2] https://www.cr0.org/progs/crctools/
[3] https://www.usenix.org/legacy/event/woot11/tech/final_files/Goodspeed.pdf
[4] http://www.phenoelit.org/irpas/docu.html
[5] http://greatscottgadgets.com/daisho

--[II. Code

e1000e driver FCS manipulation
http://dev.inversepath.com/802.3/e1000e_3.9.4-ifcs.diff

XC-2 Ethernet Kit injection tool
http://dev.inversepath.com/802.3/injector-0.1.tgz

Packet-In-Packet FCS collision helper script
http://dev.inversepath.com/802.3/tweak_packet.sh

--[III. Links

Project directory
http://dev.inversepath.com/802.3

--[EOF]--