
# Bypassing Firewalls and NATs By Exploiting Packet-in-Packet Attacks in Ethernet

Ben Seri, Gregory Vishnepolsky and Yevgeny Yusepovsky

# Introduction

Armis Labs discovered novel methods to exploit Packet-in-Packet attacks in Ethernet cables. These methods can be used under certain conditions to bypass perimeter security devices such as firewalls and NATs. Bypassing these defenses can allow attackers to mount various attacks:

1. Penetrate networks directly from the Internet
2. Penetrate internal networks from a DMZ segment
3. Move laterally between various segments of internal networks

The ultimate goal of the methods we explore is to inject fully controlled Ethernet packets in internal networks. We will demonstrate how a single Ethernet packet that is injected using these methods can be used to allow an attacker to achieve various goals:

1. Establish a Man-in-the-middle position, from the Internet, on DNS and\or HTTP requests of devices in the internal network by injecting a specially crafted broadcast IPv6 Router Advertisement. This can be used, for example, to eavesdrop on corporate communications.
2. Gain full control over devices by exploiting 1-day vulnerabilities such as CDPwn and URGENT/11, which include remote-code-execution (RCE) vulnerabilities that can be triggered by a single broadcast packet within the network.

The Ethernet Packet-in-Packet attack has been explored in the past, at a Black Hat talk back in 2013, titled "Fully arbitrary 802.3 packet injection". However, the researcher back then deemed this attack impractical for various reasons. Our research presents new methods, and supporting data that indicates this attack is more practical than previously considered.

This document will offer a deep dive into the mechanisms and conditions that allow Packet-in-Packet attacks to take place in Ethernet. In addition, this document will detail the various prerequisites for the Ethernet Packet-in-Packet to be successful, and the various methods we identified in which these can be challenged.

## Who we are

Armis Labs is Armis' research team, focused on mixing and splitting the atoms that comprise the IoT devices that surround us - be it a smart personal assistant, a benign looking printer, a SCADA controller or a life-supporting device such as a hospital bedside patient monitor.

Our previous research includes:

- CDPwn - 5 Zero Day vulnerabilities in various implementations of Cisco's CDP protocol, used by a wide array of their products. The technical whitepaper for this research can be found here:
    - CDPwn - Breaking the discovery protocols of the Enterprise-of-Things
- URGENT/11 - 11 Zero Day vulnerabilities impacting VxWorks, the most widely used Real Time Operating System (RTOS). The technical whitepaper for this research can be found here:
    - URGENT/11 - Critical vulnerabilities to remotely compromise VxWorks
- BLEEDINGBIT - Two chip-level vulnerabilities in Texas Instruments BLE chips, embedded in Enterprise-grade Access Points. The technical whitepaper for this research can be found here:
    - BLEEDINGBIT - The hidden attack surface within BLE chips
- BlueBorne - An attack vector targeting devices via RCE vulnerabilities in Bluetooth stacks used by over 5.3 Billion devices. This research was comprised of 3 technical whitepapers:
    - BlueBorne - The dangers of Bluetooth implementations: Unveiling zero day vulnerabilities and security flaws in modern Bluetooth stacks
    - BlueBorne on Android - Exploiting an RCE Over the Air
    - Exploiting BlueBorne in Linux-Based IoT deices

## Motivation for bypassing NATs/Firewalls

The majority of 0-click remote-code-execution vulnerabilities require attackers some form of network adjacency to the victim device -- either direct IP routing, or even layer 2 adjacency. This includes some of the RCEs we discovered ourselves, such as URGENT/11 and CDPwn, but this is also true for other well known RCEs such as BlueKeep and EternalBlue, for example. This is simply due to the fact that the majority of 0-click RCEs are triggered by maliciously crafted packets that are sent to vulnerable services that accept packets coming from the local network. These types of packets are rarely allowed to enter the internal network, and are blocked by perimeter security devices such as firewalls and NATs.

In certain cases, RCE vulnerabilities can even be exploited with a single maliciously crafted packet. CDPwn and URGENT/11 contain vulnerabilities that are an example of this. If an attacker gains the ability to send a fully controlled packet to a victim device from beyond a network's perimeter security defenses, these types of single-packet RCEs may become reachable for an attacker on the Internet.

Many organizations put tremendous fate on their perimeter security defenses, in the hope that they will prevent the penetration of their networks, and the devices within these networks. This leads to a state where many devices are left unpatched, vulnerable to critical vulnerabilities that may be exploited by specially crafted packets sent to them within the internal network. Thus, an attacker that is able to send fully controlled packets within such networks has tremendous hazardous potential.

Understanding some of the elementary threats to the design of these perimeter security systems is what led us to look at Ethernet Packet-in-Packet attacks in more depth.

## Packet-in-Packet attacks in wireless protocols

The term Packet-in-Packet was probably first coined by Travis Goodspeed in 2011, when he discovered a way to inject fully controlled layer 2 packets in 802.15.4 (layer 2 protocol used by Zigbee) and 802.11 (Wi-Fi), given the ability to send packets with partially controlled payloads. This sounds quite surprising, and it relies on the fact that wireless transmissions are inherently unreliable, and this guarantees that bit flips would randomly occur in transmissions, and eventually, the headers of the lower layers of the packet may get corrupted.

When this happens, the receiver of such a packet can be fooled to interpret the payload of the packet as an entirely new packet, including the previously uncontrolled low-level headers. An attacker that has the ability to even partially control the payload of such packets may be able to inject fully controlled layer 2 packets.

| Preamble | Sync | Payload |
|---|---|---|
| 00 00 00 00 | a7 | 0f ... |
| 00 00 00 00 | a^ | 0f ... 00 00 00 00 a7 ... |

^ 802.15.4 Packet-in-Packet!

In the example above, *a7* is the syncword of the layer 2 protocol used by Zigbee (802.15.4), and when it gets corrupted, the receiver will continue searching for another preamble and syncword inside the packet. A crafted payload that contains these magic numbers will get interpreted from that point on as a completely new packet.

A more recent Packet-in-Packet attack was devised in 2015 for non-encrypted Wi-Fi. The paper titled *Injection Attacks on 802.11n MAC Frame Aggregation* describes a method to target the MAC frame aggregation feature of Wi-Fi access points to exploit Wi-Fi Packet-in-Packet.

Ultimately, a similar concept is used whenever Packet-in-Packet attacks are exploited in wireless protocols -- partially controlled packet payloads, together with bit flips in the air, resulting in arbitrary packet injection.

The concept of Packet-in-Packet isn't new, but it was mainly explored in wireless protocols. In our case, however, we wanted to explore methods to bypass firewalls and NATs, and these are connected to wired networks. Does it even make sense for this to work on wired protocols?

To answer this question, let's first dive deeper into the physical attributes of Ethernet.

# Packet-in-Packet attack in Ethernet

## Ethernet PHYs

The most popular Ethernet cables are copper cables that use either FastEthernet (which is 100 Mb/s) and Gigabit Ethernet. These two PHYs have very different encodings on the physical layer.

In FastEthernet, the PHY encoding uses 5 bit symbols on the wire for every 4 bits of data:

| Data | | 4B5B code |
|---|---|---|
| (Hex) | (Binary) | |
| 0 | 0000 | 11110 |
| 1 | 0001 | 01001 |
| 2 | 0010 | 10100 |
| 3 | 0011 | 10101 |
| 4 | 0100 | 01010 |
| 5 | 0101 | 01011 |
| 6 | 0110 | 01110 |
| 7 | 0111 | 01111 |

| Data | | 4B5B code |
|---|---|---|
| (Hex) | (Binary) | |
| 8 | 1000 | 10010 |
| 9 | 1001 | 10011 |
| A | 1010 | 10110 |
| B | 1011 | 10111 |
| C | 1100 | 11010 |
| D | 1101 | 11011 |
| E | 1110 | 11100 |
| F | 1111 | 11101 |

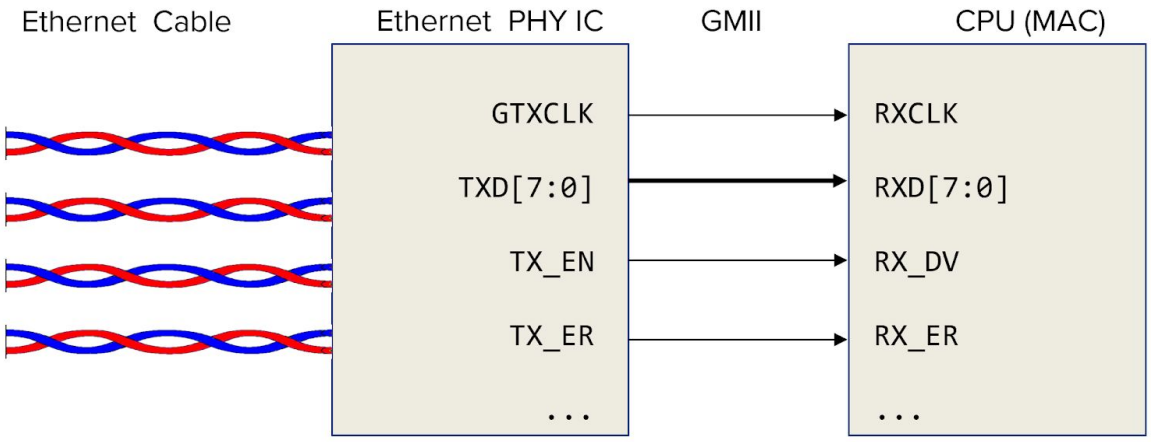
| Symbol | 4B5B code | Description |
|---|---|---|
| H | 00100 | Halt |
| I | 11111 | Idle |
| J | 11000 | Start #1 |
| K | 10001 | Start #2 |
| L | 00110 | Start #3 |
| Q | 00000 | Quiet (loss of signal) |
| R | 00111 | Reset |
| S | 11001 | Set |
| T | 01101 | End (terminate) |

4B5B Encoding table, as used by FastEthernet PHY (100Mb/s), from Wikipedia

As can be seen in the table above, the majority of the symbols are data symbols, and additional symbols are used for control -- such as *start of frame* and *end of frame*. Gigabit encoding is different, but it also uses a similar approach for delimiting frames by using start and end of frame symbols.

The important thing to understand is that there isn't any error detection mechanism at these PHY layers, other than the inherent mechanism to detect invalid symbols (not all symbols are valid). So if, for instance, a bit flip has occurred on the wire, the PHY layer will be able to detect this **only** if the corrupted symbol is now a non-valid symbol. One data symbol, for example, might get replaced with another data symbol, if a bit flip has occurred on the wire.

## Ethernet PHY/MAC Interface (MII)

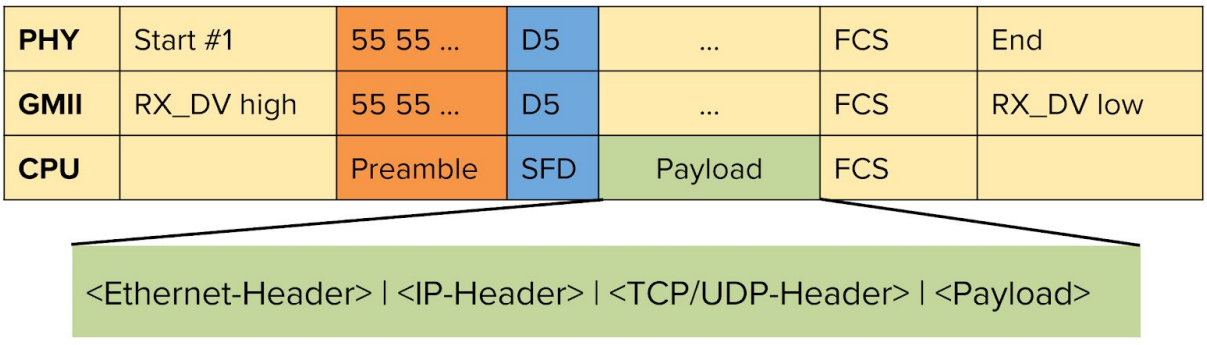



An Ethernet cable is never connected directly to a CPU or microcontroller, but rather, it's usually connected to an Ethernet PHY chip. That PHY chip is usually connected to the CPU using a hardware interface called Media Independent Interface (or modern versions of it, such as GMII or RGMII, etc.). The job of the PHY chip is to translate the Ethernet symbols from the wire, into a parallel 8-bit bus carrying our familiar layer 2 data bytes, alongside several out-of-band signals that are sent over dedicated lines - such as RX_DV (RX Data Valid signal), RX_ER (RX Error signal), etc. The RX_DV line is used for delimiting frames, and it is driven by the PHY in response to the special start and end of frame symbols encoded on the wire.

So while on the PHY layer, special start and end of frame symbols are used to delimit Ethernet frames, on the GMII bus, these signals are translated to the RX data valid signal.

## MAC layer framing

Despite the existence of framing done out-of-band by the PHY layer, the MAC layer also implements a separate framing mechanism. This mechanism is transferred in-band, on top of the framing mechanism used by the PHY layer. The familiar Ethernet frame will be prepended with preamble bytes and a *start-frame-delimiter* byte on the wire. In addition, a CRC32, called the *FCS* is appended to the end of the frame.




In the diagram above, we can see what data actually appears on the wire as part of an Ethernet frame, and how it's handled. First, a start symbol appears on the wire (called SSD - Start-Stream-Delimiter). This

is the out-of-band symbol used by the PHY layer to indicate the beginning of the frame, and the PHY chip will translate it to a high RX_DV line on the GMII bus, indicating that there's an incoming frame.

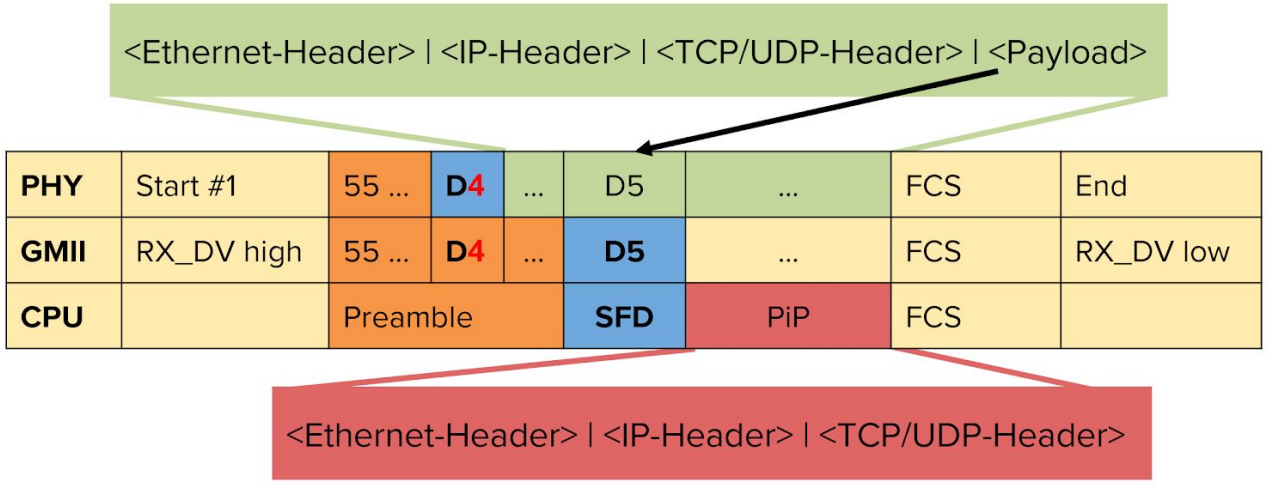
From that point on, the PHY will convert the encoded symbols to data bytes on the parallel bus. However, the first bytes of this data **aren't** actual data, but rather, they are the in-band signaling bytes used by the MAC layer to indicate the beginning of a frame. They begin with a preamble sequence (bytes with the value of 0x55) and a *start frame delimiter* byte (an SFD), that has the value of 0xD5.

When the CPU sees the RX_DV line go high it starts waiting for an SFD byte on the RXD port. All bytes before that are treated as the preamble. After these initial bytes, comes the familiar layer 2 payload, which is the Ethernet frame with headers and payload. When the *end of frame* symbol arrives on the wire, the RX_DV line goes low, and indicates to the CPU that the last 4 data bytes - are, again, not actually data, but rather, the CRC32 of the entire received frame.

Since the MAC layer responsible for the MAC framing (the preamble, SFD, and FCS), it is the one that will validate the validity of the FCS and not the PHY layer. This means that potentially corrupted frames may arrive all this way, to the CPU, without being dropped by the PHY.

## Packet-in-Packet data flow

We are now ready to connect the dots, and see why this design is susceptible to a Packet-in-Packet attack:




In the diagram above we can see the different processing stages of an Ethernet packet, in which the SFD byte got corrupted on the wire. In this case, the 0xD5 byte got corrupted and turned, for example, into a 0xD4. As noted above, from the perspective of the PHY chip, the packet starts when the start symbol (the SSD) is received on the wire, and it ends when an end symbol (the End-Stream-Delimiter, the *ESD*) is received. These symbols are not corrupted, and the PHY will use them to drive the RX_DV line. However, from the perspective of the MAC layer, on the CPU, the packet starts when the RX_DV line goes high, **and** an SFD byte is received on the RXD bus. Since the first SFD byte will be corrupted, the MAC layer will continue waiting to receive it on the bus. If the packet was specially crafted, the attacker could place a 0xD5 somewhere inside the packet's payload. If this is the first 0xD5 byte in the frame it **will** be picked up as the SFD. All the previous bytes will be considered to be the preamble sequence! In all the NIC

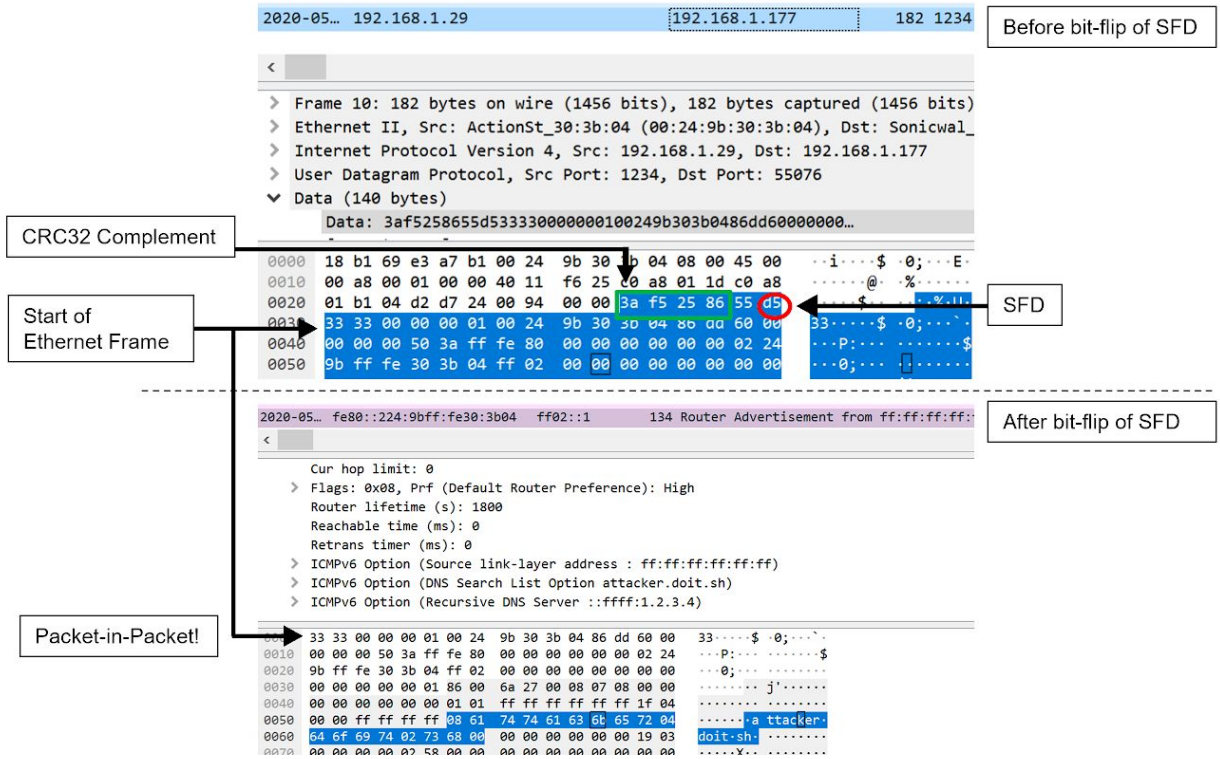
implementations we've tested, we didn't find any that limit the length of this preamble sequence, or even check its value!

As noted above, once the RX_DV line goes low, the MAC layer will know that the packet has ended, and interpret the last four bytes as the FCS (the CRC32 of the packet, starting from the SFD). So although the bit-flip of the SFD managed to move the beginning of the packet to the attacker's payload, the end of the packet, and the expected CRC of the packet cannot be controlled by the attacker.

Therefore, two conditions are required for the corruption of the SFD to turn into a Packet-in-Packet: The corrupted symbol must become some other, **valid** data symbol; The CRC32 at the end of the frame must now be correct for **both** the original packet, and the new, inner packet.

The first of these two conditions will naturally occur if bit-flips are randomly occurring on the wire. Not all corruptions of the SFD byte will lead to a valid data symbol in its place, but some of them will, and because the bit-flips occur randomly, they will eventually land on valid data symbols as well.

The second condition is more tricky. Turning a corrupted SFD byte to a Packet-in-Packet condition will only work if the attacker knows the exact contents of the headers of the original packet, so he can anticipate the FCS (the CRC32) of this packet in advance, and cause a CRC collision between the outer packet, and in the inner packet (the Packet-in-Packet). Since the attacker is the one generating the original packet, he already knows the majority of its content in advance. However, since the packet is passing a NAT, and possibly several routers inside the internal network, before the corruption takes place, the values of the source/destination MAC addresses in the Ethernet frame, and the internal destination IP are not necessarily visible to him. The attacker will have to find those out in advance (more on that in the section "Finding MAC addresses"). Once the full content of the original packet is known to the attacker, he can pre-calculate a CRC collision between both inner and outer packets, and the Packet-in-Packet condition will occur.



In the diagram above, we can see an Ethernet packet before, and after it's SFD byte was corrupted. We can see that the original packet is a UDP packet in which the attacker controls the payload, and has placed four bytes that will act as a CRC32 complement (more on that in the next section), and than a preamble byte (0x55), and a new SFD byte (0xD5). After it, begins the new Ethernet header of the injected packet. Once the SFD gets corrupted, the new packet will be re-interpreted, as shown at the bottom of the above diagram. In this instance, the attacker has chosen to inject an IPv6 Router Advertisement packet, of which he has full control.

### Calculating the CRC complement

As noted above, the CRC32 at the end of the original, outer packet, also appears at the end of the new, inner packet. Therefore, the CRC must be correct for both packets simultaneously. As it happens, CRC32 is not a cryptographic hash or anything of the sort, and is easily malleable. In order to force the CRC32 of any block of data to become anything else, a four byte complement can be calculated and appended to the data. With this in mind, the final structure of the outer packet is shown below:


The entire *outer packet*

Here, the attacker controlled portion of the outer packet begins right **after** the *outer packet header*, and before the CRC32. The attacker can choose an **arbitrary** inner packet, and then calculate its corresponding CRC normally. Then, a four byte complement can be computed, and added **before** the inner packet, such that it is **not** part of the inner packet. This complement is chosen such that it forces the CRC32 of the outer packet to be the **same** as the one of the inner packet.

A good explanation of the algorithm to force the CRC32 of any data to be any chosen value can be found here. The method essentially calculates the difference between the current CRC and the desired CRC, then calculates a polynomial power mod G(x), and calculates a reciprocal mod G(x), then calculates a product mod G(x), where G(x) is the CRC generator polynomial (as defined for CRC32).

## Possible attack payloads

The attack described in this paper requires the attacker to send lots of packets, each containing a Packet-in-Packet payload, at the highest rate possible. When these packets are sent over a cable that experiences bit-errors, a bit flip might happen on the SFD byte, causing the Packet-in-Packet condition to occur. The probability of this happening for any packet is quite low, and the attacker compensates this by sending as many packets as possible. Still, it may take a long time for the right bit flip to occur. Therefore, it's reasonable to assume that this attack can only inject a single packet in an attack that can take hours.

This effectively mandates the attack payload to be limited to a single packet. There are quite a few past examples of a single packet being enough to DoS devices, or even exploit RCE vulnerabilities. Here are a few recent examples we're familiar with:

1. CDPwn vulnerabilities in parsing of the CDP protocol by Cisco routers, switches and VoIP phones (CVE-2020-3119, CVE-2020-3111)
2. Urgent11 vulnerabilities in the TCP/IP protocol stack of the VxWorks OS, used by a large percentage of manufacturing, medical and critical infrastructure devices (CVE-2019-12256)
3. A 2018 ICMP of death exploit of all Apple MAC/iOS devices in the XNU kernel (CVE-2018-4407)

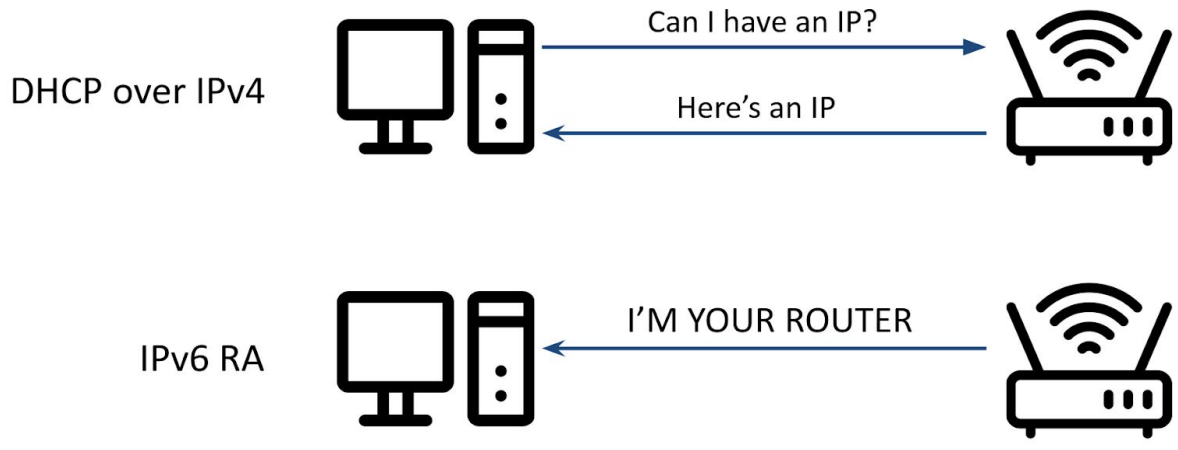
However, additional vulnerabilities are not required in order for a single injected packet to cause damage to the security of a network. Certain packets that will be handled by all standards compliant network devices are quite powerful by design.

## IPv6 Router Advertisement

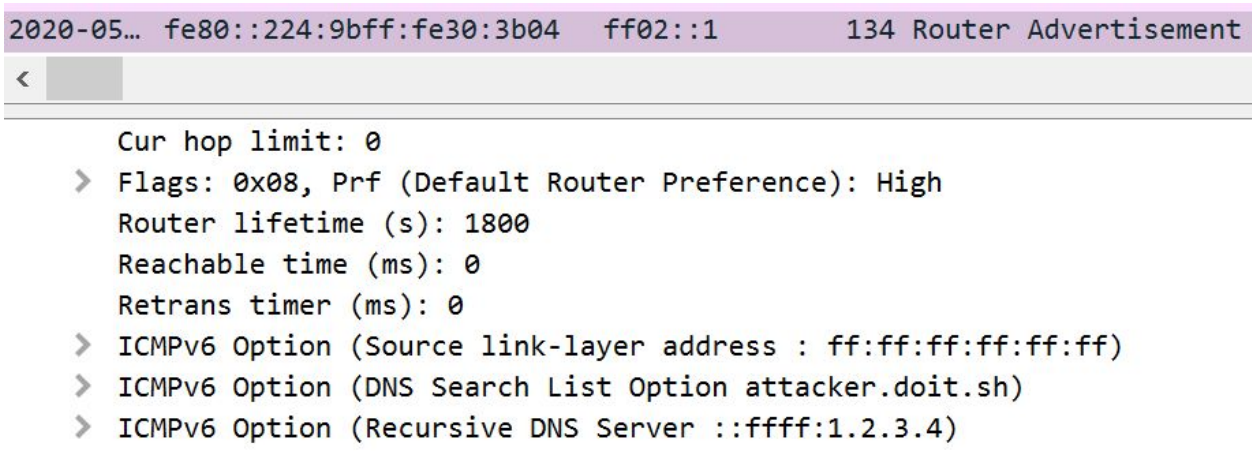
An IPv6 Router Advertisement (RA) packet is an ICMPv6 packet that notifies devices about the existence of an IPv6 router on the network segment. It is usually sent as a broadcast on every segment, and it configures IPv6 routes, DNS servers, and more, on every IPv6 enabled device that receives it.

All modern OSs support IPv6, and it is turned on by default on every network interface. An actual working IPv6 network is not required for this packet to have an effect, as obviously this is a configuration packet that's meant to set up networking in the first place.

The RA packet itself is reminiscent of the more familiar IPv4 DHCP offer packet, since it is able to configure pretty much the same settings on the receiving device. However, an important difference is that DHCP is a request-response protocol, where a client device has to first request configuration, and only then accept a response. An IPv6 RA packet can arrive unsolicited, and the receiver will apply its configuration without requesting one immediately prior. This is similar to the ancient Reverse ARP protocol that was in use before DHCP was invented.




Below is an example of such a packet setting up an attacker controlled DNS server and search domain on a target device:




## IPv6 mapped IPv4 addresses

In the above illustration, the DNS server is set to the address ::ffff:1.2.3.4 which is actually what's known as an "IPv6 mapped IPv4 address". This is a feature of IPv6 addresses, where actual IPv4 addresses are mapped **onto** the IPv6 address space, and the intent is to actually use the **IPv4 protocol** to reach them. So even though it's technically an IPv6 address being configured here, these are really **IPv4** DNS servers and will be used as such. Therefore, a working IPv6 network is not required in order to add DNS servers this way.

## Search domain and WPAD on windows

Windows Proxy Auto Discovery (WPAD) is an ancient feature of Windows, still enabled by default, which attempts to detect a configuration for HTTP proxy servers that are required by the local network. There are multiple ways for network administrators to configure WPAD, one being to set up an HTTP server on the network that would be able to answer an HTTP request to the following URL:

```
http://wpad/wpad.dat
```

Of course, "wpad" isn't a valid domain name. In this case, a "search domain" is utilised. A search domain can be configured on a network via a DHCP configuration, RA, or manually. It is basically a suffix that's appended to all unresolvable domain names, in order to attempt and resolve them again with the suffix appended. So, for example, if the search domain is set to be "attacker.com", then the above HTTP request will be effectively performed to:

```
http://wpad.attacker.com/wpad.dat
```

The wpad.dat file configures which proxy server to use for which URL. This configuration is respected by any windows application or feature that respects the OS wide proxy configuration. This includes all modern web browsers. The format of the configuration is explained here.

## Bit errors in Ethernet cables?

For this attack to work, bit-flips need to randomly occur on target Ethernet cables. In fact, the belief that these are unlikely to occur in wired cables is what led researchers in the past to dismiss this type of attack. At a Black Hat talk back in 2013, titled "Fully arbitrary 802.3 packet injection", the researcher deemed this attack impractical, due to the following reason:

*"...the reliability and extremely low error rate of wired cables make it [the Ethernet Packet-in-Packet attack] unrealistic."*

We performed a survey on Ethernet cables throughout a wide variety of organizations in which Armis operates, to find out whether this common belief is true. We were surprised to discover that imperfections in Ethernet cables are actually much more prevalent. Leveraging data from about 500K active switch ports used in enterprise networks, we found that about 1.8% of them experience a significant amount of bit-errors. According to the industry standard, the maximum allowed bit-error-rate (BER) in Gigabit Ethernet cables, for example, is one bit error in 10 **billion** bits. The high-error-rate cables identified in our survey are ones that experience a BER that is greater than this maximum BER, as defined by the industry standard. In these types of cables, a Packet-in-Packet attack **can** occur within a reasonable time (hours or less).

### Bit-error-rate in Ethernet cables - Survey results

When we looked across different segments of our install base, we noted that the rate of errors varied slightly between them:

| Vertical Segments | % of High Error Cables |
|---|---|
| Medical | 1.75% |
| OT | 1.20% |
| Other | 2.26% |
| Grand Total | 1.79% |

In Gigabit Ethernet cables that experience a high-error-rate as defined above, a bit can flip once every 10 seconds, and a Packet-in-Packet condition can then occur within hours or less.

One way for an attacker to overcome the prerequisite of bitflips randomly occurring in Ethernet cables is to simply target an organization that might use imperfect cables, and hope the benign traffic that he sends to the network traverses through one of them. An alternative method that relies on an ability to **induce** bit-flips in non-faulty cables is also something that we've explored (more on this later).

## Querying Ethernet statistics from Cisco switches

In order to locate low-quality cables that might experience significant bit-errors, it is possible to query certain managed switches for various statistics that count the number of symbol errors that are a possible side-effect from random bit-flips occurring in transmission.

In Cisco Catalyst switches these can be queried by using the following command in the Cisco shell:

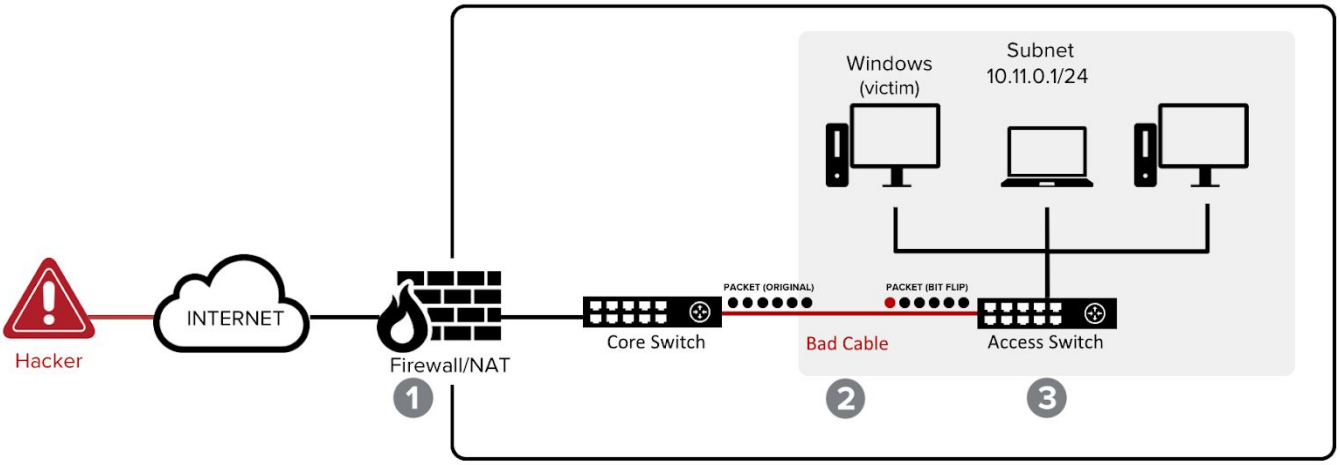
```
#show controllers ethernet-controller | inc Sym
      0 Excessive collisions              15704 Symbol error frames
      0 Excessive collisions                  0 Symbol error frames
```

Alternatively, using SNMP to query Cisco switches for these statistics is also possible via the OID named dot3StatsSymbolErrors (1.3.6.1.2.1.10.7.2.1.18).

# 1-click Attack Scenario

Now that we've outlined the various hoops that an attacker needs to jump through, to carry out a successful Ethernet Packet-in-Packet attack that can bypass firewalls/NATs, we can devise an end-to-end attack scenario. In this attack scenario, we assume that a faulty Ethernet cable exists within the network, in such a way that a significant amount of bit-flips occur on it. In addition, the attacker in this scenario has prior knowledge of the network including MAC addresses and where the faulty cables lie.

As described earlier, the Ethernet Packet-in-Packet attack works by relying on the fact that when bit-errors randomly occur in transmissions, an attacker can leverage and abuse them to inject fully controlled packets when they occur in a certain way.




*How an Ethernet Packet-in-a-Packet Attack can work*

The following outlines how such an attack could happen:

1. The attack starts by an attacker sending a link to a user inside a target network that leads to an attacker-controlled website:
    a. The browser on the user machine will use WebRTC to initiate a UDP connection to the attacker's server. This is not a vulnerability, but rather a built-in feature within browsers to allow UDP communications for a variety of streaming applications.
    b. The outgoing UDP packet from the browser will be registered by the NAT as a new UDP "connection", meaning the NAT will now allow incoming packets from the attacker to traverse back to the victim device. Since UDP is a connectionless protocol, the NAT will allow these incoming packets to traverse, until a timeout passes **from the last received or sent packet.** Meaning the attacker can now send a stream of benign UDP packets to the victim device that will encapsulate the Packet-in-Packet payload, from the Internet. The stream can continue even after the victim closes the browser!
2. The stream of benign UDP packets will traverse through the faulty cable, and some of these will experience bit-flips. Eventually, the SFD byte of one of these packets will bit-flip, and that will enable the Packet-in-Packet condition. The attacker will craft the payload of these UDP packets as described in a previous section - a CRC32 complement will be added to the payload so the FCS of both inner and outer packets will collide, and a new preamble and SFD bytes will be placed before the inner packet.
3. When the Packet-in-Packet condition occurs, the Ethernet controller on the receiving end will interpret the payload of the packet (that is attacker-controlled) as an entirely new packet. This will allow the attacker to control low-level headers of injected packets - essentially sending fully controlled Ethernet frames. These types of packets would have been otherwise blocked by firewall/NAT solutions.

Once the attacker is able to inject an Ethernet packet to the internal network, he can send, for example, any of the packets described in the section "Potential attack payloads". These can lead to the takeover of vulnerable devices, or as illustrated in the diagram above, a Windows device can be the target of the aforementioned IPv6 Router Advertisement packet. As described earlier, this packet can lead the attacker to a Man-in-The-Middle position on DNS and HTTP traffic of Windows devices – by registering a Search Domain and abusing the Windows Proxy Auto Discovery (WPAD) feature. It is important to note that both IPv6 and WPAD are enabled by default in Windows devices (even in an IPv4 network).

A video of this demonstration is available here.

# Physical layer of Ethernet

An Ethernet cable, usually Cat 5e or 6 copper cable, consists of four pairs of wires. While only two pairs are used in FastEthernet (100 Mb/s), Gigabit Ethernet and above requires all four pairs. Each symbol is transformed into a voltage difference between the two wires in a pair (differential pair).

While a differential signal accommodates for ground differences, and the reference voltage drops along lengthy cables, there is an additional challenge of a coexistence in a real life environment. Due to the fact that a current may be induced in any wire with external electromagnetic radiation, and any wire which passes either power or a signal also emits such radiation (Maxwell's laws), the electrical devices/cables may potentially interfere with each other. This phenomenon is called electromagnetic interference (EMI). In the case of the cable, an EMI might be either external to the cable or induced by the pairs inside the cable from one to another (crosstalk).

The coexistence of electrical devices and cables is mitigated by electromagnetic compatibility (EMC) standards and regulations, required for any consumer equipment. Those standards and regulations ensure both the immunity and low enough emission levels for any equipment. In the case of an Ethernet cable, the minimal required level of EMC is achieved by a twisted pair wiring technique.

Illustration of a twisted pair

A twisted pair is a type of wiring where both wires in the differential pair are twisted together. As a result, the induced current by electromagnetic emission is almost identical on both wires (The similar signal on both wires is called a common mode, and the differential signal is called a differential mode. Using this terminology, each signal is a sum of the common and the differential modes). I.e - Only a very small part of the EMI results in a differential induced current.

The subtraction of the two voltages in a differential pair, cancels out the common mode. Thus, a twisted pair is susceptible almost solely to the small part of the differentially induced EMI. In addition, the opposite signal in the two wires of the differential pair ensures that the interference created by each is mostly canceled out by the other. This way, the twisted pair wiring greatly improves EMC, by reducing electromagnetic radiation and crosstalk, and improving rejection of external electromagnetic interference.

## Shielding

In addition to the twisted pair geometry of wiring, the electromagnetic compatibility can be further increased by a shield, which is an electrically conductive barrier, that significantly attenuates both the emission and the induced current from EMI. The shield may be applied to each of the twisted pairs individually, an outer shielding for the whole cable, or both. An outer cable shielding provides both the

protection from external sources and attenuates the emitted radiation, while the shielding for each of the pairs provides additional crosstalk protection.

The information about cable shielding is standardized by ISO/IEC 11801:2002 (Annex E) and uses the following abbreviations: an x/yTP shielding stands for x type of external shield for the whole cable, and a y shielding for each of the twisted pairs (TP in abbreviation). The three most common types are the unshielded cable (UTP), the cable in which there is only an outer layer of shielding (F/UTP), and the cable in which each of the twisted pairs is shielded (U/FTP, which is sometimes mistakenly referred by cable manufacturers as STP).

Different shielding used by different types of cables

## Types of Ethernet cables

In addition to the shielding abbreviation, the ISO/IEC 11801 also standardized several cabling categories which differ in a number of parameters, including the maximum frequency, electromagnetic compatibility, maximum length, and testing procedure. This standard defines the most commonly used Ethernet cables, from the lower-end of Cat5 to the Cat6 and above.

The original Cat5, was the lowest-end Ethernet cable, for up to 100Base-TX Ethernet, with a length of up to 100 meters. It was defined by ANSI/TIA/EIA-568-A and later by ISO/IEC 11801, IEC 61156, and EN 50173. It was deprecated in 2001 and superseded by Cat5e, which is now the most commonly used cable. The Cat5e standard guarantees lower crosstalk than the Cat5, and communication for up to Gigabit Ethernet speeds. Whereas the Cat5 was UTP, Cat5e cable can optionally have foil on either the whole cable or each single twisted pair. Thus, it comes in either UTP, F/UTP, or U/FTP.

Cat6 cable ISO/IEC 11801 2nd Ed. (2002), ANSI/TIA 568-B.2-1, extend the supported Ethernet speeds to up to 10G-BASE-T, but, with only a 55 meters limit for the 10Gbit Ethernet.

In addition to the maximal distance and bandwidth, the standards for each cable define the maximum attenuation, impedance range, propagation delay, electromagnetic characteristics, operating temperatures, NEXT (near end crosstalk, which is the most basic type of crosstalk and the most relevant for this discussion), and more. The most important specifications are summarized in the table below:

| | Cat5 | Cat5e | Cat6 |
|---|---|---|---|
| **Frequency** | 100MHz | 100MHz | 250MHz |
| **Attenuation (min. at 100MHz)** | 22dB | 22dB | 19.8dB |
| **Characteristic impedance** | $100\Omega \pm 15\%$ | $100\Omega \pm 15\%$ | $100\Omega \pm 15\%$ |
| **NEXT (min. at100 MHz)** | 32.3dB | 35.3dB | 43.3dB |
| **Return Loss (min. at 100MHz)** | 16dB | 20.1dB | 20.1dB |
| **Delay Skew (max. at 100m)** | N/A | 45ns | 45ns |
| **Supported Networks** | 100BASE-T | 1000BASE-T | 100BASE-T, 1000BASE-TX |

### Differential noise margin

As an example, 100Base-TX uses 3 differential voltages -1V, 0V, 1V (See source [1]). The Cat5 and Cat5e cables have 22dB attenuation for 100m (See sources [2], [3]), whereas Cat6 has a bit lower attenuation. The Cat5e cable is defined to work for up to 100 meters. The receiver should deal with an up to 22dB attenuation with an "acceptable BER", This also defines the spec for the minimal voltage level for an Ethernet signal, near the receiver:

$$V_{min} = 10^{(20*log(1[V]) - 22)/20} \approx 80\ mV$$

This means that a constellation of three voltage levels of -80mV, 0mV +80mV can be met in real life conditions. At those conditions, the noise margin cannot be higher than 40mV, and should be ensured by EMC.

## Possible reasons for bit errors in an Ethernet cable

A non zero BER is very common in communication systems. When valid equipment is properly used, The BER can be kept very low by the EMI standardizations and the EMC regulations. However, there are several cases in which the BER may rise above the acceptable threshold:

1. Cable damage that causes an excessive attenuation
2. Cable mismatch that causes a return loss, and interference between the current signal and previously returned one
3. A susceptibility to EMI due to impedance mismatch (explained below), incorrect shielding, or shield damage
4. Crosstalk between the Ethernet pairs, or even an additional, close cable
5. An EMI that is higher than what is allowed to be created by consumer electronics, with an Electromagnetic compatibility (EMC) license

[1] David A. Weston (2001). Electromagnetic Compatibility: principles and applications. CRC Press. pp.240–242. ISBN 0-8247-8889-3. Retrieved June 11, 2011

[2] https://www.farnell.com/datasheets/2762432.pdf

[3] https://www.cablek.com/technical-reference/cat-5---5e--6--6a---7--8-standards

The following sections detail each of these cases.

## Excessive attenuation

An excessive attenuation is almost exclusively a result of some degradation to the conductor. In an operating environment, such degradations are usually caused by cable stress through rolling, bending in a single point, torsion, or a random flexing. In addition to the obvious damage from stress beyond the elastic region of the conductor, even repeated stress, below the point of 15% elongation can cause permanent damage and an attenuation increase (See source for the above points [4]). Similar damage can also be caused by the shear and torsion stress, or a high weight put on the cable (for example - someone steps on the cable), but not as common as the flex damage. It is important to mention that such point damage may cause a wire to be almost torn or even pass the signal only when the cable is bent in specific ways.

Another very common reason for conductor fatigue that causes excessive attenuation is a stress in the termination point (i.e connector). This can happen from bending, vibrations, momentum from a cable weight on unsupported connectors, and poorly crimped or soldered connectors.

In the case of a sole excessive attenuation, the total attenuation is a sum of all attenuations (in dB) along the way (Ignoring impedance mismatch). In the case of a valid cable, the attenuation is given by the multiplication of length by the attenuation per unit length (22dB per 100m in Cat5e). Any additional attention should be added to this number, which may cause the total attenuation to exceed the maximum allowed 22dB, for a 100-meter cable, even for a much shorter cable.

## Impedance mismatch influence on the signal propagation

An impedance of the cable is governed by the geometry of the differential pair. Any change of the geometry along the cable, including point damage due to mechanical stress, shield degradation, a change in the twists, and the distance between the pairs in the cable, causes a mismatch in the impedance. This affects cable performance in two ways:

1. Any mismatch causes a return loss (i.e - part of the signal, returns to the source. Thus, the remaining transmitted signal strength is lessened). As it reduces signal strength, it may be treated as a part of an overall attenuation.
2. Two mismatches or more, cause interference, in which the signal returned from a mismatch point, reaches the previous mismatch point, and is reflected yet again. Then, it interferes with the original signal. This effect will cause a variation in the logical level of the original signal and, for this discussion, may be treated as a noise, which further reduces the noise margin.

## EMI susceptibility

Any electrical device or equipment, emits some level of electromagnetic radiation. The cable is protected from the interference of this radiation by the twisted pair geometry and sometimes a shield. Damage to either of the two may cause the cable to be inappropriately susceptible to EMI. A shield

[4] Moll, Kenneth W. and McCarter, David R., W. L. Gore & Associates, Inc., Flex Life in Cables, Electronic Packaging and Production, June 1976, p. 29-30, 3435.

degradation is especially common, and may happen due to aging, UV exposure (See source[5]), humidity, harsh temperature conditions, and mechanical damage (See source[6]). In addition, mismatch points are excessively susceptible to external EMI.

The EMI causes an induced current, which may be treated as noise with a given voltage level. An excessive susceptibility to EMI would result in a higher noise level from the same EMI source. If the level of the noise surpasses the calculated noise margin of the cable of 40mV, a bit flip will occur.

## Crosstalk

Crosstalk is a special case of EMI susceptibility (and emission), in which the EMI is between two digital signals. The most common type of crosstalk is crosstalk between separate differential pairs in the same cable, called the near end crosstalk (NEXT). Whereas an STP cable has a shield for each pair to reduce the amount of NEXT, A UTP and F/UTP or FTP cables lack this protection, and thereby are more susceptible. The minimal requirement for crosstalk protection is specified in each Ethernet cable category, however, any damage to the cable, which causes either a mismatch, or a shield degradation, may increase the level of crosstalk above the acceptable level. In FastEthernet, the effect of a NEXT, for our discussion, can be treated as additional noise.

A cable carrying 1000BASE-T is not susceptible to NEXT, but it is susceptible to crosstalk caused by a pair from another nearby Ethernet cable, called Alien Crosstalk (AXT). This type of crosstalk is very common, due to a large number of cables commonly running closely and in parallel to each other. Both the outer shield of F/UTP, and the shield for each pair in the STP cable may give some level of protection from AXT. The best protection, however, is achieved with a cable that has both an outer shield and a separate shield for each pair.

## Excessive EMI

The EMI can cause a degradation in Ethernet communication performance not only because of excessive cable susceptibility to the electromagnetic radiation, but also by an excessive EMI from an external source that exceeds the EMC regulation. Such a source may be a device or equipment, creating a high EMI, which can cause an unacceptable BER, even in a perfectly undamaged cable. One such example is a proximity to high voltage lines. (See source[7])

## Cable measurements setup

Two types of equipment were used for cable measurements: a Fluke Cable Qualification Tester with a dedicated model for Ethernet cable analysis, and an Electrical Network Analyser.

In the case of a network analyser, the differential pairs were connected directly to the network analyser ports with an Ethernet connector, soldered directly to an SMA connector. Such setup creates a mismatch, due to the 50-ohm impedance of the network, and 100-ohm impedance of the differential pairs. Despite the seemingly large impedance mismatch, this simplistic setup, as explained below, provides enough precision for the purpose of our research.

[5] https://www.elandcables.com/the-cable-lab/faqs/faq-what-are-the-main-causes-of-electrical-cable-failure
[6] https://www.gore.com/resources/tech-note-understanding-cable-stress-and-failure-high-flex-applications
[7] https://www.nexans.com/US/files/DCCC03040901R1.pdf

This setup has two mismatch points, one is near the transmitting port, and the other is at the end of the cable. The former mismatch point gives the following return loss:

$$S_{11-near-end-mismatch} = 10 * log\left(|\Gamma|^2\right) = 20 * log\left(\left|\frac{Z-Z_0}{Z+Z_0}\right|\right) = 20 * log\left(\frac{1}{3}\right) = -9.5\ dB$$

Whereas, the further mismatch point, for a cable of at least 20 meter, gives:

$$S_{11-far-end-mismatch} \approx 2 * (-22 * 20/100) - 9. = -18.3dB$$

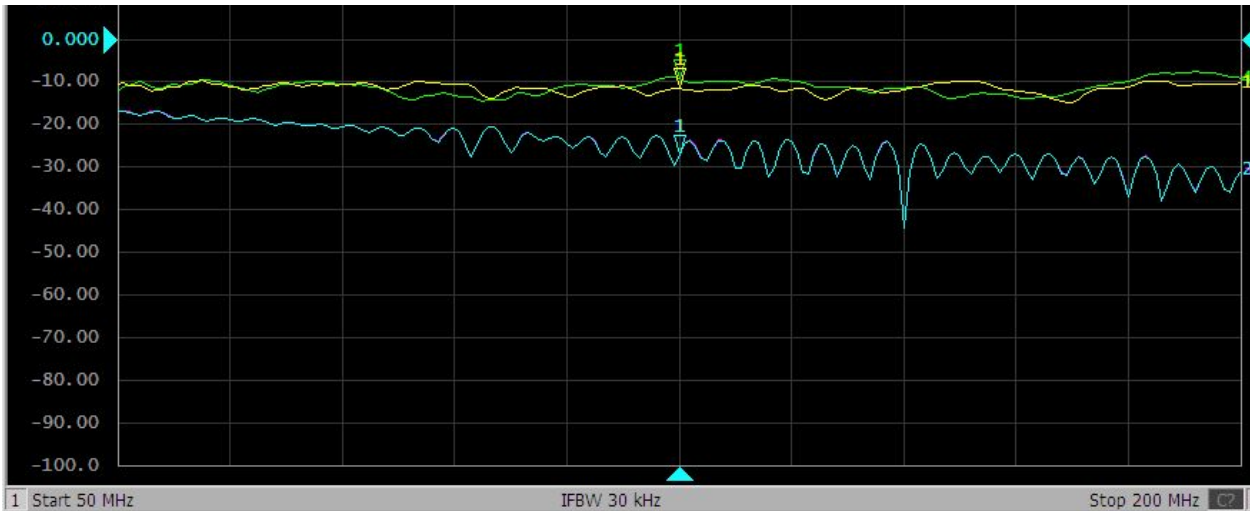
The far end of the cable gives quite negligible return loss, and is close in its value to an average measurement precision with a network analyzer. Moreover, it is not far from the 20dB return loss of a typical Cat5e cable. For this reason, for a long enough cable, the measurement can be treated, with a good approximation, as a single mismatch point.

For the return loss, the 9.5 dB defines the maximal precision. However, in 100BASE-TX Ethernet, each pair is either TX or RX, which renders the transmission loss as almost the sole, relevant parameter:

$$S_{21-near-end} = 10 * log\left(|\tau|^2\right) = 10 * log\left(1 - |\Gamma|^2\right) = -0.51\ dB$$

Thus, for measurements with an accuracy of less than 1dB, this setup may give good enough precision.

A measurement of a transmission loss (blue), and the return loss from each terminal (yellow and green) of a valid 60 meters cable, in our setup:

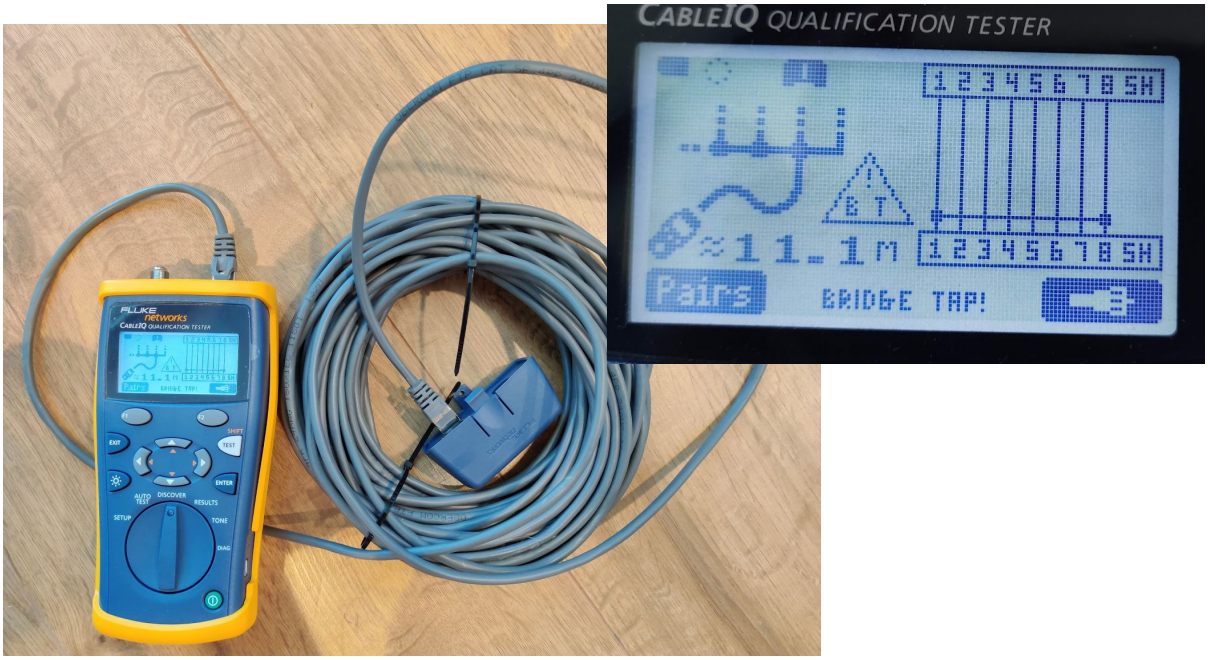



*Network Analyzer screenshot. Frequency range is between 50MHz and 200MHz*

## Detecting cabling faults with a tester

Portable devices, such as the Fluke Cable Qualification Tester can be used in order to detect and classify cabling faults. This was used around the offices of the authors of this paper, in order to find what turned out to be a small number of misbehaving cables.

The first step to finding such cables in real world networks is to query the statistics of switch ports, on all managed network switches that support such queries (as described in the section "Querying Ethernet statistics from Cisco switches"). Then, knowing which ports have elevated error rates, it's possible to physically approach the potentially faulty cables and test them individually.




Testing a particularly faulty cable

In the case above, a cable that has a short between 2 wires was located! This particular specimen works only at FastEthernet speeds, since otherwise it does not have 4 working pairs. For FastEthernet, only 2 are enough. Therefore in this example, the 1-2 pair is active, and the 7-8 pair is unused. The short thus causes wire 8 to act as an "antenna" connected to the 1-2 pair, causing reflections and interference.

Since auto-negotiation is usually enabled, the switch port falls back to FastEthernet when this cable is connected, as the initial attempt to use Gigabit fails. This cable has very high error rates, in the range of $1/10^7$.

## Lab reproduction of cabling faults

One of the objectives of this paper is to allow the reader to understand and reproduce the Packet-in-Packet attack over faulty Ethernet cables so this attack can be further researched and understood. However, it would be unreasonable to expect the reader to seek out real-world faulty cables in order to do this. Therefore, below are a few methods that we've determined to be a reliable way to simulate such faulty cables. These methods do not attempt to reproduce realistic faults. That is, these faults are extreme both in the sense that they're not likely to occur naturally, and also in the sense that they cause a very high BER. On the other hand, the faults are not excessively bad, such that the cable ceases to work entirely. Regardless, the **effects** caused by these faults are essentially the same as with real world faulty cables, therefore these reproductions are good enough to use for testing purposes.

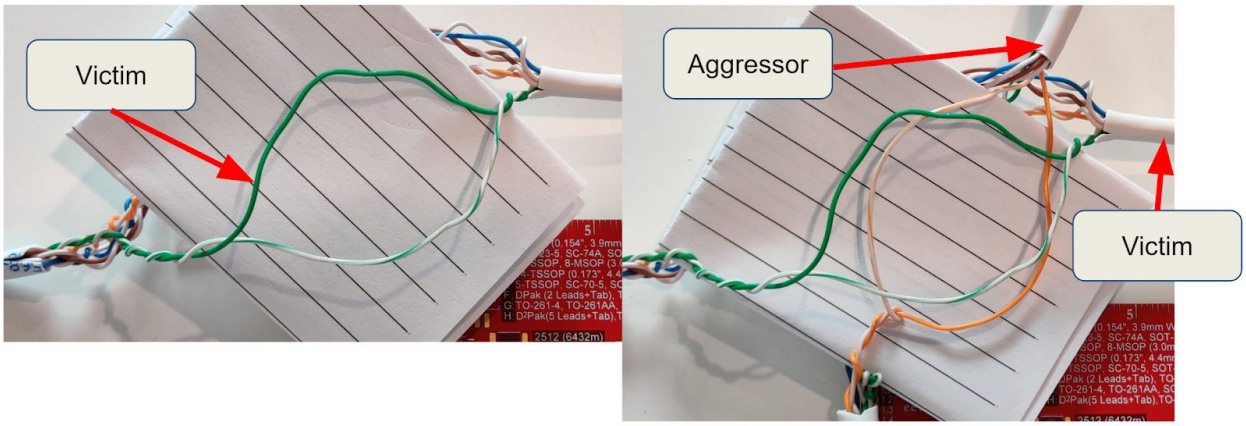
## The crosstalk model

The first method relies on deliberately corrupting 2 separate cables, and placing them close to each other. This will cause an extreme AXT (alien crosstalk) between the two.

Specifically, the steps to create these are such:

1. Remove about 10cm of the shield from two different Ethernet cables
1. Untwist one pair out of each cable
2. Open the untwisted pairs into a loop, and bring the two loops close to one another
    a. The closer they are the higher the BER
3. Both cables will interfere with each other as long as both are connected to active Gigabit Ethernet ports on both sides.

The picture below illustrates the two loops created in our setup:




Logically, one cable is considered the *aggressor* which causes the interference, and the other is the *victim* which experiences it. The aggressor doesn't need any actual Ethernet frames traversing it - the idle signal between 2 active ports is good enough.

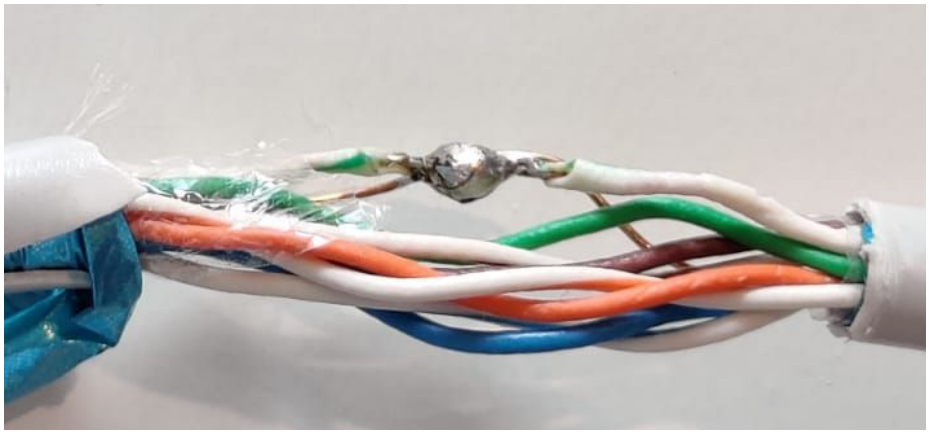
In the example of the setup above, the AXT crosstalk had turned out to be about 20dB between the 2 pairs! That's similar to the attenuation of a long 90m Cat 5e cable! This can be seen in the screenshot below:

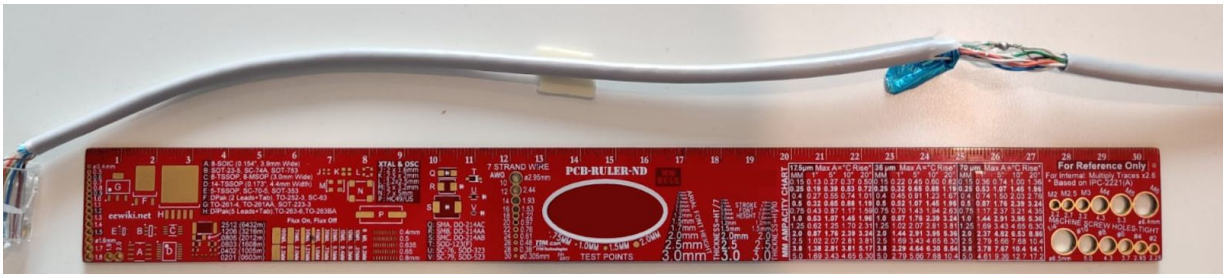

As such, if the reproduction *victim* cable is connected in series (discussed below) to a long cable that already introduces attenuation, the injected noise is guaranteed to be as strong as the signal -- thereby causing the desired bit flips.

## The short model

A more common failure mode of Ethernet cables is due to an internal short of one of the wires to the grounded shield of the cable. We tested this type of failure while using the cable with a Gigabit Ethernet configuration. Such a failure should still allow a high-speed transmission, but will cause significant degradation.

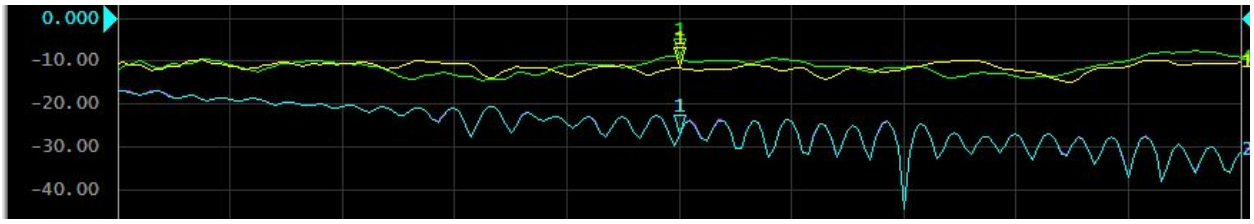
The intentionally created short to ground, in our setup, is close to the victim's receiver terminal. As in the previous example, this cable is connected in series with a longer cable (60m in the case of our setup) in order to add extra attenuation.



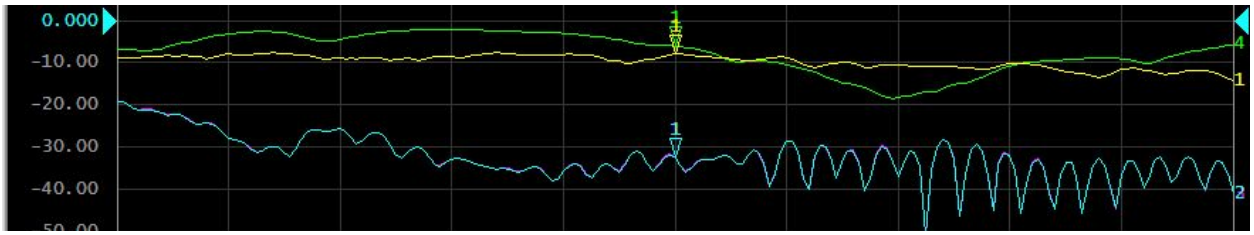



The short to ground causes several different influences on the cable. It is an impedance mismatch that causes signal reflection, it causes an attenuation due to a pull to the ground, it emits EMI and it is also susceptible to EMI. Due to the complexity of the phenomenon, the significance of this effect may depend on many parameters, including the exact number of wavelengths the short lies from the terminal. In our example, the short to ground caused an additional attenuation of about 10dB, and increased EMI susceptibility.

The following network analyser screenshot shows the cable attenuation for a 60m cable (in blue), before the pair was shorted to the shield:




Whereas, below is a screenshot after the short to the ground was added:

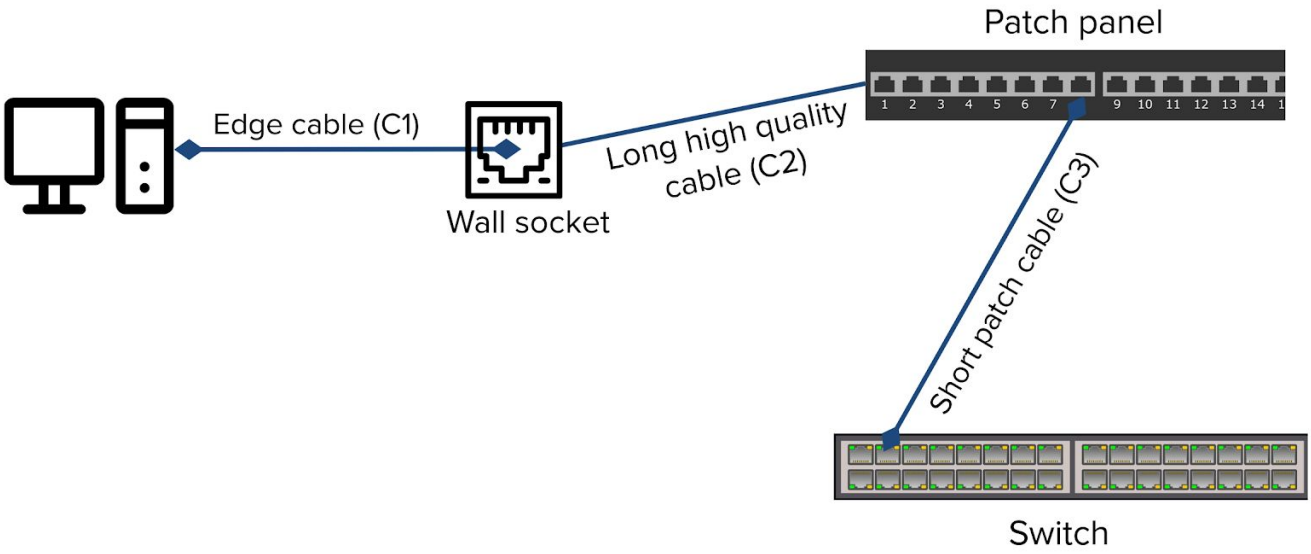



A high level of reflections and significantly increased attenuation is observable in this second shot.

## Model scenario for cables connected in series

So far we've analyzed and referred to Ethernet cables according to well known parameters and specifications. However, for the purposes of the Packet-in-Packet attack described in this paper, a "cable" is just an abstraction. In practice, a "cable" is whatever physically connects 2 active Ethernet ports one to the other. In reality, this is **rarely** an actual single cable.

It is very common, in fact, for cables to be connected **in series** via wall sockets and patch panels. The most common scenario for connecting endpoints would be something similar to this:




In the illustrated case, an endpoint device is connected via an edge cable (C1) to a wall socket. The wall socket is an RJ45 female socket, which internally wires the attached cable to another cable, C2, which is a potentially long cable that runs through the walls, and into a comms room. There, the cable run is wired into a patch panel, where another RJ45 female socket accepts a patch cable (C3) that finally connects to the Ethernet switch.

This entire series of interconnections actually consists of 3 cables, 4 RJ45 jacks, and 2 RJ45 sockets. Since all of this is connected **passively in series**, it can simply be seen as a single long "cable", for the purposes of the discussed attack. A fault in any of these components can be seen as a fault in this single long "cable".

Impedances add up in series. Therefore, if C2 is a long, yet high quality cable, it will still contribute a significant amount of impedance to the entire cable run. If, for example, C1 is a faulty cable -- the receiver on the endpoint will effectively "see" a **long faulty cable** attached to it. High impedance means a smaller differential noise margin (as explained earlier), and a fault means that weaker interference can impact the cable. Together, these factors result in an elevated bit error rate (BER) for the receiver, allowing the Packet-in-Packet attack.

Connections between 2 on-site network appliances (such as between 2 network switches), follow a similar pattern. There will be 2 patch panels involved, in 2 separate comms rooms. Therefore, the scenario described above is probably the most common way devices are wired on a network.

This makes it easier to see why in a small, yet significant percentage of cases, the BER on a switch port is much higher than allowed by the standard.

## Primitives for a 0-click attack

In the "1-click attack scenario" section above, an attack that required user interaction was described. The user needed to visit an attacker controlled web page, so that a UDP packet would be sent through their firewall/NAT to an attacker controlled server. This was done in order for the attacker to know a UDP IP/port 4-tuple that's allowed through the firewall, and that the NAT would translate to the user's internal address. This is simply how "outgoing connections" work. While UDP is a **connectionless** protocol, there is still a concept of a "connection" when it comes to firewalls/NATs. The existence of a connection is simply determined by a unique address/port 4-tuple and a **timeout** from the last packet that was sent as part of the "connection" (a few minutes, usually).

Without knowing an IP/port 4-tuple that the NAT would direct to the victim's internal address, an attacker could never send any packets that would reach the victim over the faulty cable. In the 1-click scenario, the attacker **created** a new 4-tuple with a known value, since the attacker's server was simply the other side of the connection. However, a new connection is not required, if the following conditions are met:

- The attacker is able to guess an already existing 4-tuple that the NAT already directs to the victim.
- The attacker is able to spoof IPv4 source addresses in packets sent over the Internet, in order to create packets with that same 4-tuple

### Spoofing IPv4 source addresses on the Internet

While most VPS/cloud providers do not allow outgoing IPv4 packets with a spoofed source address, this is usually a limitation imposed by the provider itself, or their ISP.

Naturally, this means that some providers still allow this capability, either because they don't care, or because they cater to certain clients whose DDoS attacks depend on being able to spoof source addresses. For example, SSDP DDoS attacks.

In 2020, it's no longer very easy to just stumble upon such a provider. However, if you limit your search to Russian providers, and look for reviews that talk about this "feature", you can still find some, although they won't advertise themselves as such. We've found and used one that is a fairly serious VPS provider, and is operating for more than 10 years. As far as we know, they have never disallowed IP spoofing for outgoing traffic.

In order to test if your VPS is capable of spoofing, simply try to send an IPv4/UDP packet with a spoofed source address using **scapy** to another VPS (from a different provider), and see if it arrives.

## Google DNS 4-tuples

Armed with the ability to spoof source IPs on the Internet, the attacker now needs to guess a 4-tuple that's allowed through the firewall and will be directed to the victim by the NAT.

DNS is a request-response protocol that uses UDP packets sent to port 53. On a given network, all devices are usually configured to use the same DNS servers. As far as we know, Google DNS, on the IP address 8.8.8.8 is the most popular DNS resolver on the Internet. Some networks do not use an Internal DNS resolver, but rather, set 8.8.8.8 to be the resolver for every device on the network directly. Additionally, this address is sometimes hard coded to be the DNS resolver of some devices. As a result, every DNS request out of such a network will have a 4-tuple with predictable components:

- Dest IP is 8.8.8.8, source IP is the external address of the network (considered to be known)
- Dest port is always 53
- The only unknown part is a 16-bit source port

While this does mean that there can only be about 64K concurrent DNS requests on such a network, that's not a real problem, since functionally, DNS request-response "connections" are very short. Less than half a second in pretty much all cases. For firewalls/NATs, however, there is an additional problem. They only see DNS traffic as generic UDP packets, they are not aware that these packets are DNS. Therefore, as far as they're concerned, each such "connection" will last a few **minutes** before it times out. Luckily, this is also not a functional problem, because when all 64K source ports are exhausted by these "concurrent connections", the NAT in question simply throws out the oldest ones for the sake of new ones to be established.

In this case, if an attacker simply iterates through all possible source ports, they can send UDP packets to random machines on the internal network! The only condition is that those machines have performed some DNS requests to 8.8.8.8 in the last few minutes.

| Source | Destination | SrcPort | DstPort | Info |
|---|---|---|---|---|
| 8.8.8.8 | 31. [redacted] | 53 | 10290 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10292 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10293 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10294 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10295 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10296 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10297 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10298 | Standard query 0x0000 |
| 8.8.8.8 | 31. [redacted] | 53 | 10299 | Standard query 0x0000 |

Spoofed packets from 8.8.8.8 sent to victim NAT to all Dest Ports

| Source | Destination | SrcPort | DstPort | Info |
|---|---|---|---|---|
| 8.8.8.8 | 192.168.2.173 | 53 | 10093 | Standard query 0x0000 |
| 8.8.8.8 | 192.168.2.51 | 53 | 10161 | Standard query 0x0000 |
| 8.8.8.8 | 192.168.1.217 | 53 | 10188 | Standard query 0x0000 |
| 8.8.8.8 | 192.168.1.217 | 53 | 10188 | Standard query 0x0000 |
| 8.8.8.8 | 192.168.2.69 | 53 | 10295 | Standard query 0x0000 |
| 8.8.8.8 | 192.168.2.0 | 53 | 10307 | Standard query 0x0000 |
| 8.8.8.8 | 192.168.2.130 | 53 | 10341 | Standard query 0x0000 |
| 8.8.8.8 | 192.168.2.89 | 53 | 10355 | Standard query 0x0000 |

Some of those traverse the NAT to random Dest IPs on the LAN!

Most conveniently for the attacker, when those spoofed UDP packets do traverse the NAT/firewall, they are considered to be valid traffic of each of those UDP "connections", thereby **preventing** the timeout from closing the connection. Therefore, an attacker that keeps iterating overall source ports will only increase their throughput over time, until **all** possible DNS 4-tuples become used for sending attacker packets into the internal network.

**It is important to see that this method is not entirely useful** for our Packet-in-Packet attack, since the packets will be sent to **random** destinations on the internal network. The attacker must choose a **single** internal destination in advance, as they have to know the MACs in the packet headers. Those MACs will obviously be different in all those random destinations. Knowing which source port is the "right" one to reach only the desired destination is not possible as long as there is **no feedback** to the attacker. We didn't find any method to receive such feedback without additional prerequisites.

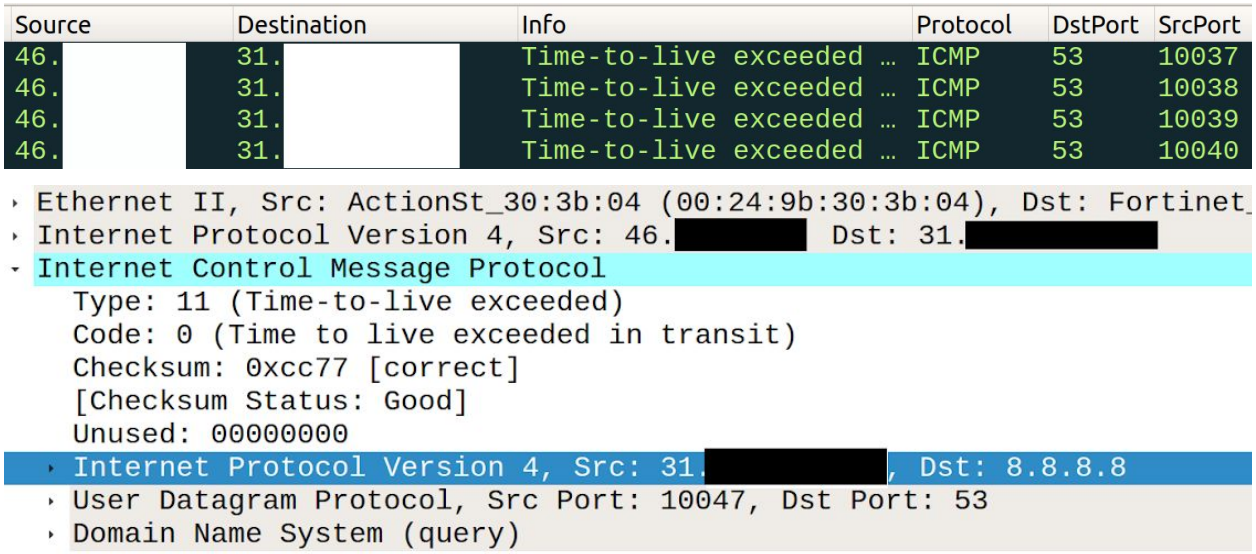
It **is possible**, however, to receive this feedback in a proximity attack scenario involving sniffing encrypted Wi-Fi packets, as will be discussed in a later section.

## Alternative method: ICMP errors

There is an additional primitive we've found that enables us to perform the above attack **without** needing a VPS provider that allows IP spoofing on the Internet.

The spoofed packets discussed above traverse the firewall/NAT since they are part of an ESTABLISHED "connection". However, there can be "RELATED" packets as well. For example, for every UDP "connection", there are ICMP error packets that can be RELATED to it. A simple case is when sending a UDP packet to a closed port. This results in an "ICMP destination unreachable packet", sent back to the client from the destination host. The NAT/firewall is **aware** of the ICMP protocol, and will direct those packets to the correct internal host.

ICMP error packets always **include** the packet headers of the packet that caused the error. This is where the NAT/firewall takes the information from about which UDP/TCP 4-tuple this ICMP error packet is RELATED to.




Some ICMP errors, like the one above, may arrive from an **intermediate router**, rather than the destination host. The above packet is a "TTL exceeded" error, that can arrive from any router between the client and destination server. Therefore its source IP address can be any valid address! The IP address used to identify the RELATED 4-tuple of this ICMP error is taken from the **internal** IP header that's included in the ICMP error payload!

Therefore, instead of sending spoofed DNS responses to the victim's external IP address, The attacker will send "ICMP TTL exceeded" packets that are RELATED to the guessed DNS 4-tuple. This means that spoofing the source IP of those packets is no longer needed!

However, a downside of this method is that RELATED packets won't keep the UDP "connection" alive as far as the NAT/firewall is concerned, and it **will** time out after a few minutes.

## Finding MAC addresses

As mentioned previously, the attacker must know the MAC addresses that appear in the layer 2 headers of the original packet that's meant to traverse the faulty cable. One of these MAC addresses is the MAC of the destination device (the victim), and the other is of the nearest router to the victim. On smaller networks, that router is usually also the NAT/firewall device that's on the edge of the network.

There is no way, that we know of, to learn those MAC addresses from the Internet directly, or via some web browser based primitive. However, MAC addresses are not considered to be a secret by any threat model, and they were not designed as such.

In some cases, it's fairly easy to guess them. For example, take a non-Internet attack scenario. An attacker is inside a DMZ, within zero hops from the firewall. This attacker is part of the same layer 2 network as the WAN interface of the firewall device. Therefore, the attacker knows the MAC address of this interface. This is very useful to the attacker, since as is the case with most network equipment, the MAC addresses of all the interfaces of the same equipment are adjacent. This is also the case with most

firewall appliances. Therefore, it's easy to guess the MACs of the other interfaces of the firewall by simply knowing one.

## Discovering MACs from Wi-Fi monitor mode

A very practical approach to discovering the internal MAC addresses inside a network is via Wi-Fi sniffing, when the wireless access points of the target network are bridged to the wired network. Even on encrypted WPA2 networks, the MAC addresses appear in plain text over the air. Therefore any attacker in physical proximity to the network can sniff them using commodity hardware.

Since the APs are bridged to the wired network, the MACs in the air are exactly the same MACs that appear on the wired network, inside the Ethernet frames that are exchanged with the wireless APs.

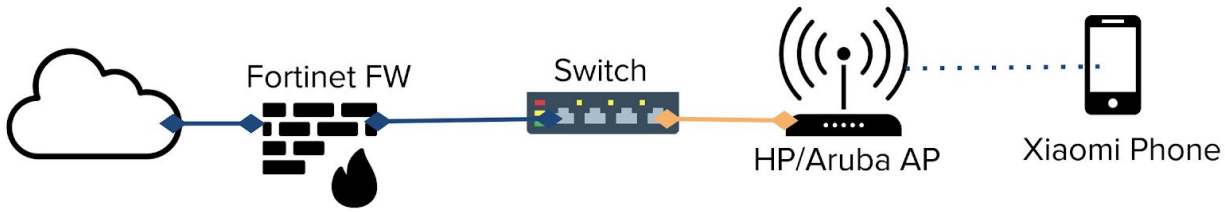
```
Fortinet_[redacted]    XiaomiCo_b2:33:37    QoS Data, SN=2110, FN=0, Flags=.p....F.C
Fortinet_[redacted]    XiaomiCo_b2:33:37    QoS Data, SN=2111, FN=0, Flags=.p....F.C
```

Sniffed 802.11 data frames

```
Receiver address: XiaomiCo_b2:33:37 (48:2c:a0:b2:33:37)
Transmitter address: HewlettP_cd:c9:e0 (44:48:c1:cd:c9:e0)
Destination address: XiaomiCo_b2:33:37 (48:2c:a0:b2:33:37)
Source address: Fortinet_[redacted]
BSS Id: HewlettP_cd:c9:e0 (44:48:c1:cd:c9:e0)
STA address: XiaomiCo_b2:33:37 (48:2c:a0:b2:33:37)
```

5 MAC addresses appear in each frame in plain text

In the screenshot above, we have five MACs for every 802.11 frame, of which one is **always** the MAC of the wireless device, and another is **always** the MAC of another device on the network -- be it wired **or** wireless.




In the case above we have a Xiaomi phone talking to a Fortinet firewall, which is its default gateway to the Internet. These are the two MACs that appear in any Ethernet packet coming from the firewall to the device when traveling **over the wired network** on the way to the access point.

While this scenario requires physical proximity to the network in order to learn the MAC addresses, this only needs to occur once. An attacker that has arrived on site and logged all visible MAC addresses in the air, can use them in any later attack since MAC addresses never change.

## Discovering allowed traffic through the firewall using WiFi sniffing

In the previous sections 2 important primitives were presented:

1. Sending packets from the Internet to random devices behind a firewall, by guessing predictable UDP 4-tuples that were recently allowed through the firewall
2. Ability to sniff MAC addresses from Wi-Fi by being in physical proximity to the target network/device

The latter primitive can actually be expanded a bit, as WPA2 encrypted data frames reveal more than just the plaintext MACs of the devices -- they also reveal the length of the plaintext packets. This is simply because the length of the plaintext packets matches the length of the encrypted packets.

Armed with these 3 primitives, it's now possible to devise a method to send packets from the Internet to a **particular** device behind a firewall, given the following conditions:

1. The target device uses Wi-Fi to connect to its internal network.
2. The target device uses Google's DNS resolver (8.8.8.8).
3. The attacker is in physical proximity to the target's Wi-Fi AP.

Given the above conditions, the attacker will be able to learn the MACs of both the target device, and its closest router, using the Wi-Fi monitor mode. To be clear, this information will be enough to later perform a Packet-in-Packet attack on any cable **on the way** from the router to the target AP.

The method works as follows:

1. The attacker will be sending spoofed packets from 8.8.8.8 over the Internet to the external IP of the network.
    a. The attacker controls the dest port (16 bit) and the length of every packet.
    b. The only variable in the UDP 4-tuple of these packets is the dest port, the rest are fixed and known.
2. The attacker creates a histogram of packet lengths normally seen in the air, and chooses the length that has the lowest occurrence. The distribution of packet lengths on a network is not random, and therefore good candidates are guaranteed to exist.
3. The attacker sends a spoofed packet of this length to **every possible dest port** to the external IP of the network, iteratively, in a loop.
4. The encrypted version of **some** of these packets, with this exact length, will appear in the air, and should be detectable over time.
5. The attacker then **halves** the dest port range, and starts iterating over that.
    a. If the correct dest port is in the range, the packets of the chosen length will become **more** detectable above the background noise, since there are now fewer possible ports overall, and therefore more packets hit the correct port.
    b. If the correct dest port is not in the range, the chosen length packets will not be detectable over background noise

6. The attacker will continue this in a binary search pattern, until the correlation between dest ports and wireless device MACs is established.

At this point, the attacker will be able to send benign UDP packets, with known MAC addresses and controlled payloads over the Ethernet cables leading from the router to the AP. By continuously sending these packets, the attacker will also cause the NAT/firewall to keep this UDP 4-tuple alive forever, allowing the attack to take as long as necessary.

## Proximity attack using an EMP device

The method described in the previous section basically allows an attacker in proximity to a Wi-Fi AP to send benign UDP packets, with a controlled payload, over the Ethernet cables **leading** to that AP.

If one of those cables proved to be faulty, this would make for a practical attack. However, most cables are not faulty, and the attacker has no way to know which is which in advance.

Faulty cables, as defined earlier in this paper, are cables that are susceptible to reasonable electromagnetic interference. However, what about unreasonable interference? An unshielded cable, even if not faulty in itself, given an already attenuated signal, should become susceptible at higher interference levels.

Since physical proximity is already part of the requirements for this version of the attack, the attacker might as well bring extra equipment in order to artificially induce powerful interference, such that it would affect even non-faulty cables.

This sort of equipment would most likely be referred to as an EMP weapon. These devices are usually designed to kill electronic circuits at a distance. They operate by sending very short but very powerful wideband bursts of radio, commonly in the 100MHz to 2GHz range. The **wavelengths** in this frequency range just about match the **lengths** of wiring inside various equipment. At high enough power levels, any such wire effectively becomes an antenna, such that it will apply voltages to random parts of circuits... With high enough power, this is meant to permanently destroy electronics.

In the case of our attack, we don't need to destroy anything. All that's required is to interfere with already attenuated signals inside unshielded Ethernet cables. The only protection against interference in such cables is the twisted pair geometry itself, which is in practice not enough for high EMI levels.

### Prior research on "EMP simulation" devices

The correct search keywords for finding information on building high power wideband interference sources are "EMP simulation", as this is the only justified use for building such devices outside of military uses. Some modern equipment is designed to be protected from high power EMI interference, and therefore needs to be tested under those conditions. Therefore, "simulation" devices are built-in academia and certain private sector entities for this purpose.

Of course, there isn't any difference between a "simulation" device and a real one. Below are a few references to papers describing the design of various components of such devices:

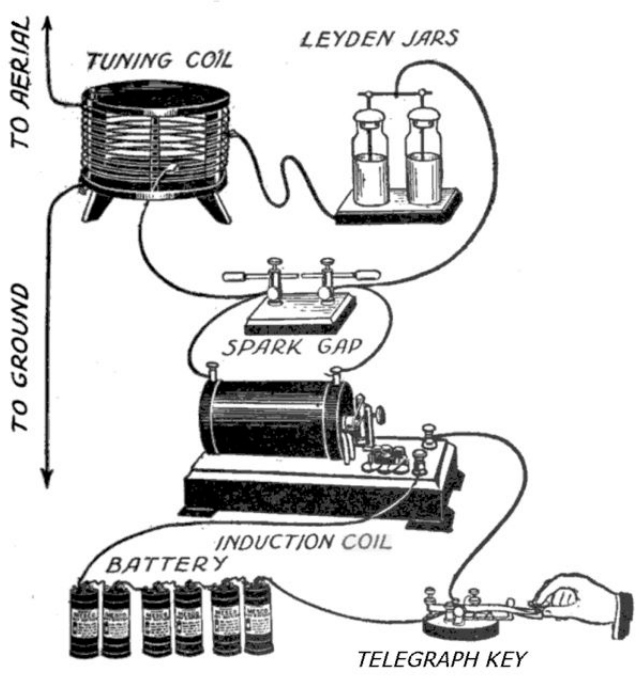
1. A Peaking Switch to Generate a High Voltage Pulse of Sub-nanosecond Rise Time [2012]
2. Self contained source based on an innovating resonant transformer and an oil peaking switch [2011]
3. An oil peaking switch to drive a dipole antenna for wideband applications
4. Generation of sub-nanosecond pulses using peaking capacitor [2016]
5. Impulse Electromagnetic Interference Generator [2004]
6. A 500Kv pulser with fast rise time for EMP simulation [2013]
7. Analysis of half TEM horn antenna for high power UWB system [2017]

These papers mostly describe a very similar device. A low inductance capacitor is charged to about 500kV, then it is discharged **very quickly** via a fast spark gap (in transformer oil or compressed gas) in parallel to a wideband antenna. The **rise time** of the discharge current is below 1 nanosecond, resulting in a wideband burst of radio up to GHz frequencies. This cycle is repeated at tens or hundreds of Hz, in order to maximize the odds of inducing damaging voltages in victim circuits.

The power levels described are indeed very high, but for the purposes of our research, should not be required.

## Wideband interference generation using a spark-gap radio transmitter

The earliest type of radio transmitter, invented in the late 19th century, isn't in fact very different from the description above.

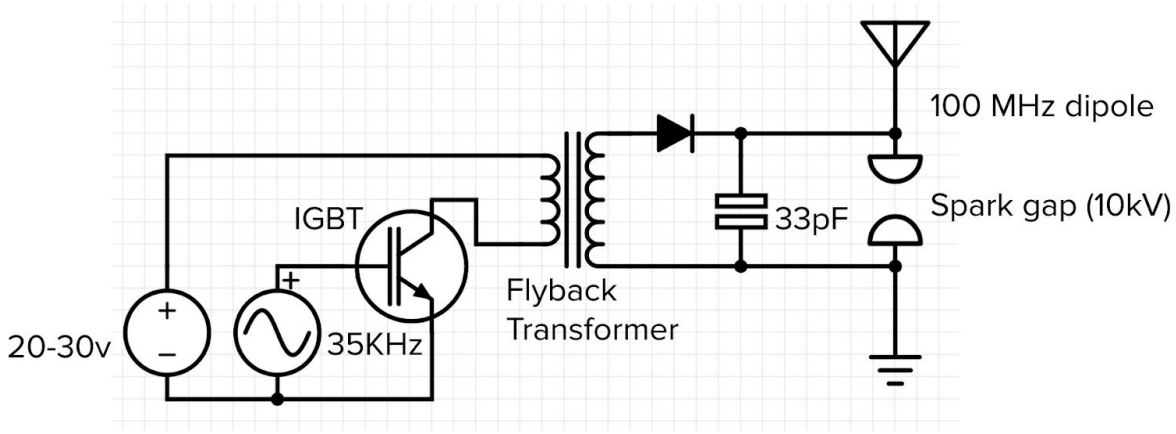



Spark-gap radio transmitter, from Wikipedia

A similar design can be used in order to create a controlled experiment device that will work as an interference source at a distance of a couple of meters from an unshielded Cat6 cable.

**Disclaimer: Building such a transmitter today is quite easy, yet extremely dangerous. The voltages used can kill a person instantly, and capacitors will store those voltages even when the device is turned off. No one should attempt to build this device without prior high voltage experience.**

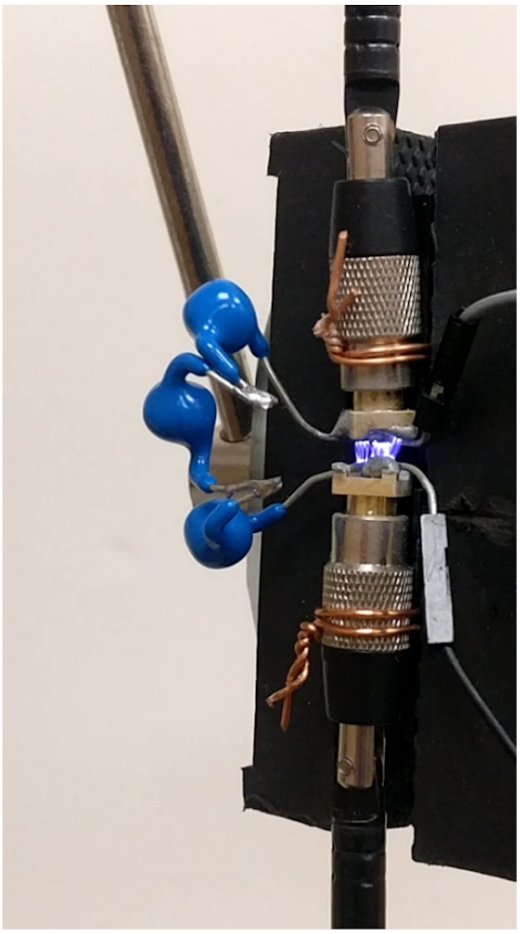
In order to build such a device today, off the shelf components can be used to create a 20kV DC power supply. The RF parts of the circuit are simply high voltage capacitors that can be found on Digikey, and a custom made antenna. The spark gap can be a simple open-air gap, which will have a rise time of 5-10ns (as observed experimentally by us). This will create powerful wideband noise in the 50MHz-150MHz range.




The more current that the power supply can provide, the higher the rate of discharges will be, since the capacitor will be charged faster in every cycle.

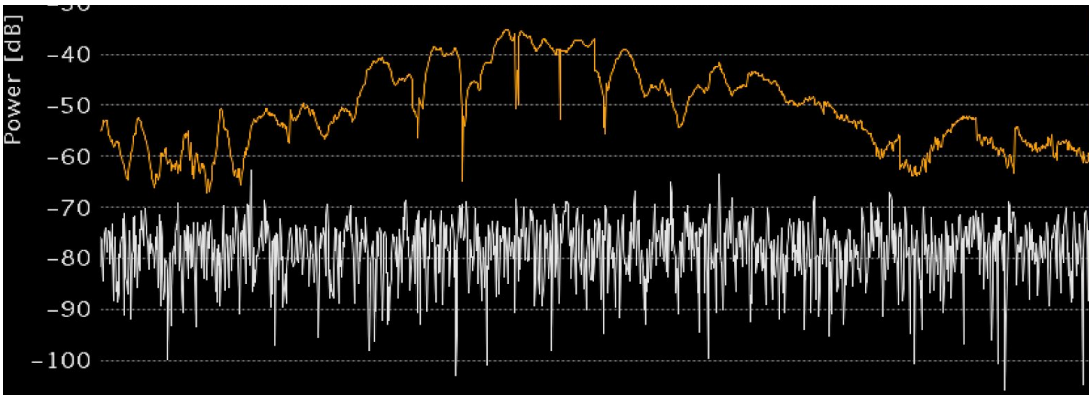
If the rate of discharges becomes too fast, the air between the spark gap electrodes can become continuously ionized, creating a plasma that does not disappear between discharges, but rather acts as a dead short preventing the capacitor from charging in the first place. This must be prevented by keeping the rate slow enough, limiting the RF output power.

Modulating the on/off time of the power supply can help with keeping the air from becoming continuously ionized. This allows increasing the individual power of each pulse, by lowering pulse the repetition rate.

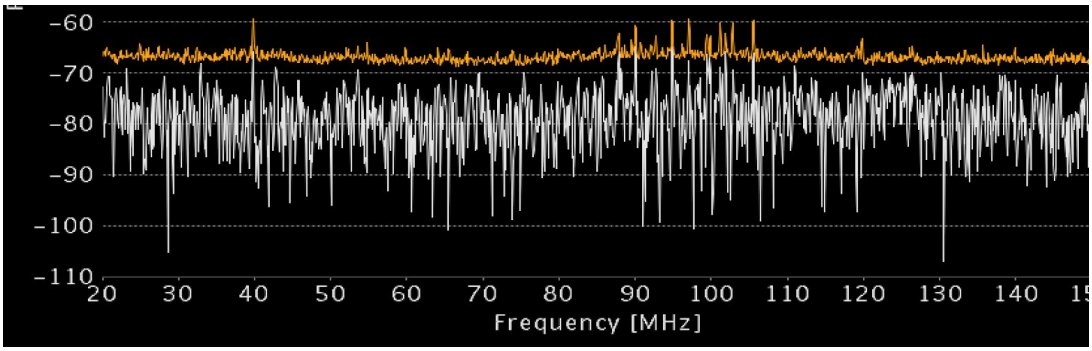



**On the left**: The controlled experimental setup of the EMP device. **On the right**: Spark gap in operation. The setup is located in an underground fortified room acting as a makeshift Faraday cage

The transmission spectrum of this EMP device is shown below:




Transmitter turned on



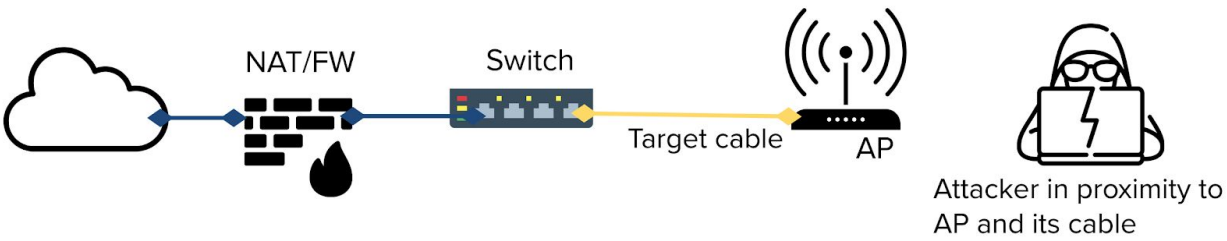

Background spectrum

The power levels shown can be disregarded, as the wideband pulses are very short (mere nanoseconds), the repetition rate isn't very high (1 kHz or so), and these spectrum analyzer graphs show power averaged over time.

Experimentally, this device was powerful enough to cause a very high error rate on an unshielded 10m long Cat6 cable in the same room with the device. Analysis of the transmission power and effect on cables will be elaborated upon later.

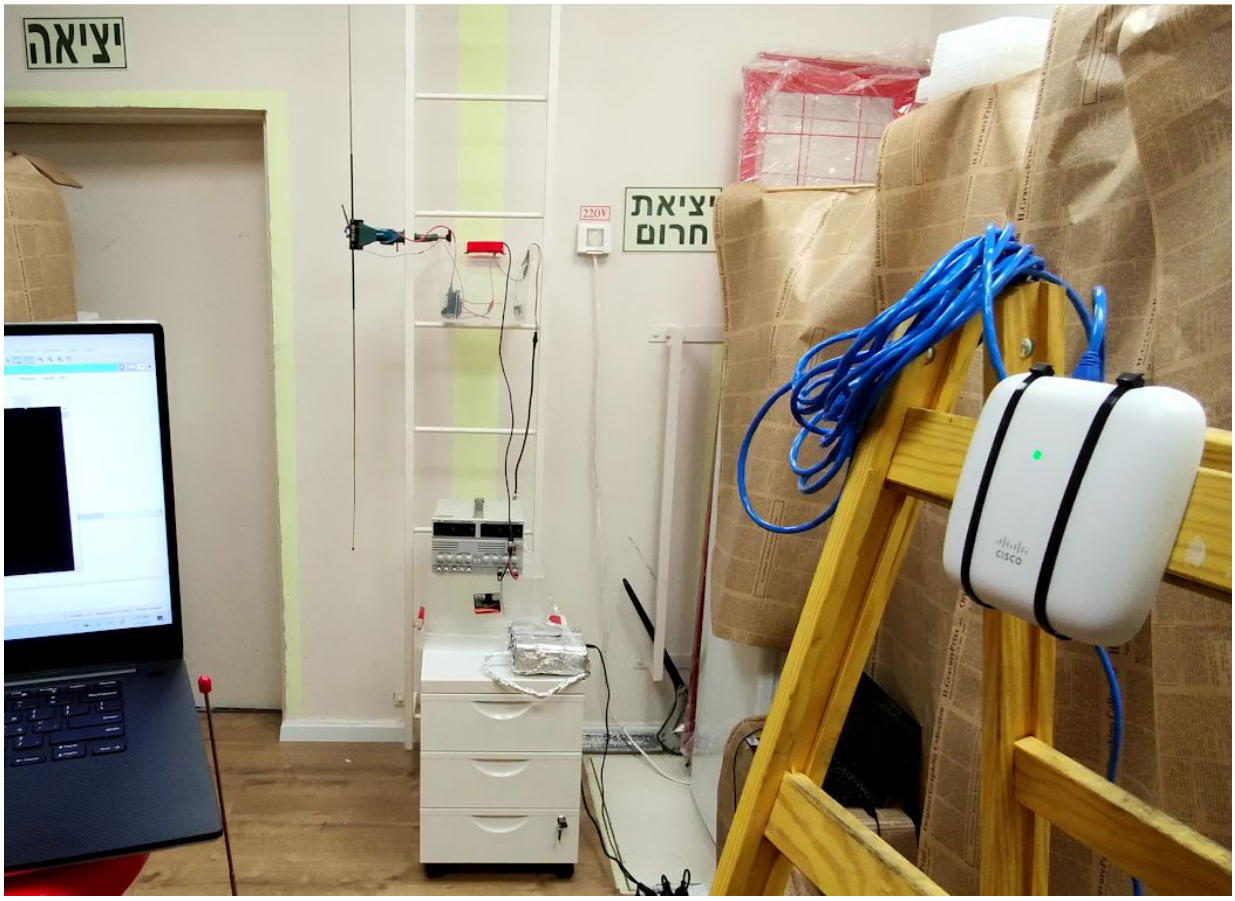
## Attack model and experimental setup

Using all the attack primitives that were discussed in this paper in previous sections, it's now possible to devise an end-to-end proximity attack. The model for our experiment is as follows:



The attacker is physically located in proximity to the Wi-Fi AP, such that they can sniff encrypted packets via monitor mode, and also be located near the AP's Ethernet cable. The attacker also has a server on the Internet, and is able to send packets to the external IP of the target network, hitting the firewall.

In our experiment, the EMP device is located 2.5m from the target cable, as shown in the following photograph of our experiment:




*Left: victim machine associated with AP.* *Right: AP and 10m Cat6 UTP cable*

The target cable in this scenario is the 10m Cat6 UTP cable (blue cable in the setup), which is located in the test room. The cable is then connected to a wall socket, which then **attaches this cable in series** with a 60m shielded Cat 7 cable that runs through the walls up to the nearest switch. Therefore, the signal inside the target UTP cable is already attenuated. Attaching cables "in series" by using wall sockets and patch panels was discussed in the "Model scenario for cables connected in series" section.

As in all previous scenarios in this paper, the purpose of the attack is to inject fully controlled Ethernet packets into the target network. In this case, these will be injected downstream from the AP's cable, therefore affecting the AP and all its associated devices.

The steps to perform this attack are as follows:

1. The attacker arrives on site and approaches a candidate Wi-Fi AP that's connected via an unshielded cable. The attacker gets physically close to the cable on the AP's end.
2. The attacker listens in Wi-Fi monitor mode, and collects the plaintext MACs of potential target wireless devices and their closest router.
3. The attacker begins sending spoofed UDP packets from their Internet server to a predictable UDP 4-tuple on the external Internet IP address of the target network, such as spoofed Google DNS response packets (as discussed in a previous section)
4. The attacker then looks at the lengths of the encrypted 802.11 frames in the air, and uses the algorithm described in the section "Wi-Fi sniffing assisted discovery of allowed firewall 4-tuples" in order to find a single correct UDP 4-tuple that is directed by the NAT to one of the target devices that are currently visible in the air.
    a. At this point, the attacker can now send benign UDP packets, with controlled payloads, to a device behind the Wi-Fi AP. Additionally, the MACs are now known.
5. Then, the attacker begins sending lots of UDP packets with the Packet-in-Packet payload to the discovered UDP 4-tuple, at high throughput, from the Internet.
6. In parallel, the attacker powers on the EMP device, and waits for a bit flip to occur on the unshielded cable. The EMP is continuously sending out pulses at a high rate.
7. Once a bit flip occurs on the SFD byte of an Ethernet frame, the Packet-in-Packet condition is achieved, and a fully controlled packet is injected!
8. Since to the AP this newly injected packet is not distinguishable from any other packet from the wired network, it will **encrypt it and send it** wirelessly to all associated devices.
9. If the injected packet was an IPv6 RA, it will now configure the DNS, routing and search domain of those devices.

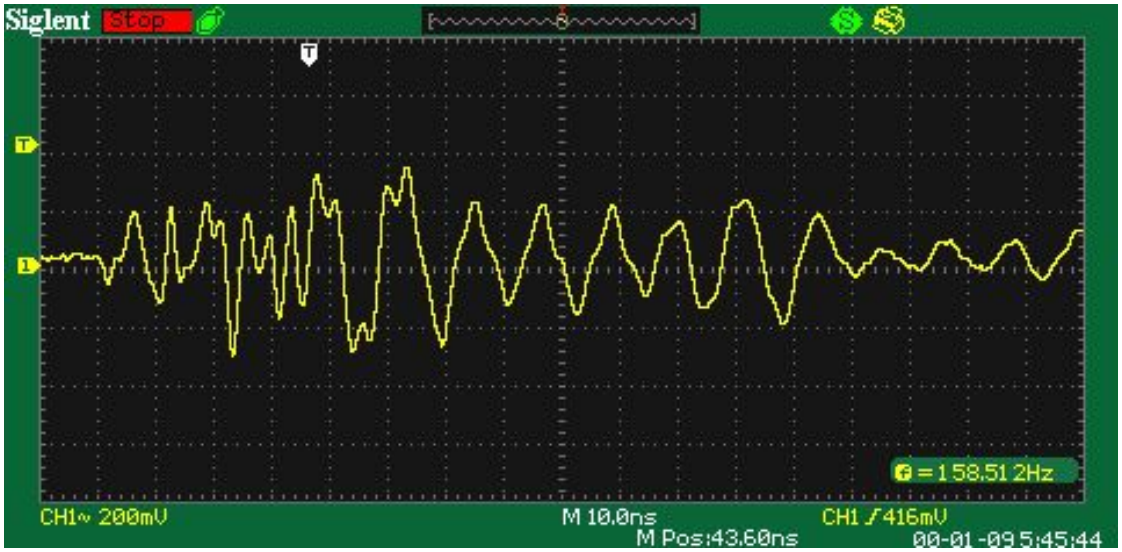
Interestingly enough, the Wi-Fi signal doesn't appear to be much affected by the EMP interference in our experiment! Mainly because the output of the described EMP device is centered around the 100MHz range, and the Wi-Fi AP operates above 5GHz. In any case, 802.11 (Wi-Fi) actually **has** a retransmission mechanism for layer 2 packets, therefore even if the Wi-Fi signal is affected along with the signal on the Ethernet cable, the injected packet will be **retransmitted** on the Wi-Fi channel as many times as needed for it to arrive!

This is indeed a fairly complex attack, however, all of these steps were performed in a lab setting successfully. A video of this demonstration is available here.

## EMP pulse measurements

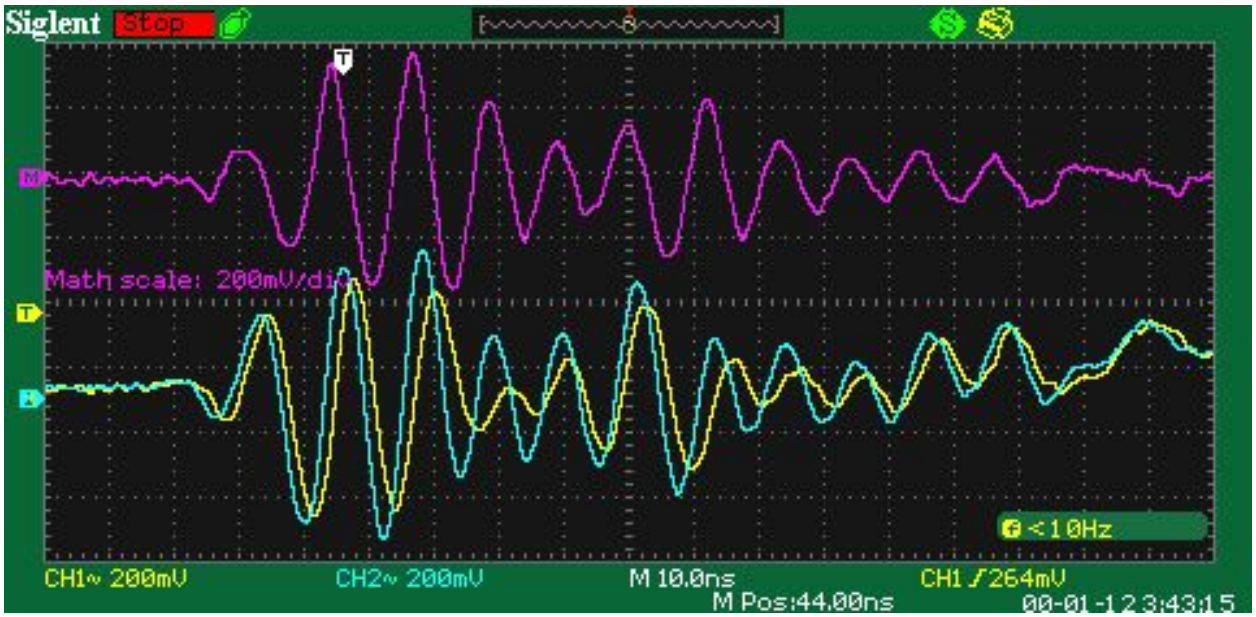
To try and better understand the effect of the EMP device on UTP cables, certain limited measurements were performed. First, an oscilloscope probe loop was placed 2.5 meters from the device:

Standard oscilloscope loop shorted to its own ground, exposing a single unshielded loop




The result is visible above. The device induces voltage pulses in the loop of wire, such that they correspond to the main frequency (80MHz) shown in the spectrum analyzer screenshots. The waveform here gets up to 600mV peak-to-peak. As a reminder, the noise margin at the receiver of a long Ethernet cable can be as low as 40mV. Of course, this is the noise margin for the **differential** signal, and the above represents the level of common mode interference.

Another experiment was performed, where the 2 wires of a twisted pair inside of a 10m Cat6 UTP cable were connected to 2 separate oscilloscope channels. The math function of the scope was utilized in order to see the differential signal by subtracting the 2 voltage levels.




Here too, the EMP pulse is visible, however, it is different enough between the 2 wires of the pair such that it causes a differential signal as well.

# Conclusion

The most surprising element of this research is that network performance issues, such as the amount of bit-errors experienced on a network's cables might also pose a security risk to the network itself. This requires the development of mitigations in network infrastructure devices that deal with this specific attack, but also offer better ways to detect edge cases that might arise in Ethernet communications.

As always, further research is required. For instance, our controlled experiment, using an EMP device to induce bit errors on non-faulty Ethernet cables, was not tested to the extent of this capability. We do not yet know whether this capability can become a dangerous and effective weapon, if perfected, and what is the maximum distance in which this attack can be successful. A better understanding of how EMI attacks can impact Ethernet cables is far from complete.

Our research was able to challenge many of the prerequisites that are required for a successful Ethernet Packet-in-Packet attack to take place. However, future research might be able to tackle some of these limitations in different ways, that might make this type of attack a much more accessible, and dangerous threat. Advancing defense capabilities in network infrastructure devices, and preventing this attack from becoming a threat is a challenge worth pursuing.