**Abstract**

In this paper we present three attacks on private internal networks behind a NAT and a corresponding new protection mechanism, Internal Network Policy, to mitigate a wide range of attacks that penetrate internal networks behind a ANT. In the attack scenario, a victim is tricked to visit the attacker's website, that contains a malicious script that lets the attacker access the victim's internal network in different ways, including opening a port in the NAT, or sending a sophisticated request to local devices. The first attack utilize DNS rebinding in a particular way, while the two others demonstrate different methods of attacking the network, based on application security vulnerabilities.

Following the attacks, we provide a new browser security policy, Internal Network Policy (INP), which protects against these types of vulnerabilities and attacks. This policy, is implemented in the browser just like Same Origin Policy (SOP), and prevents malicious accesses to internal resources by external entities.

**Introduction**

A large class of malicious exploits on the internet trick users to transmit an undesired request to a website that trusts them, CSRF (Cross Site Request Forgery) is an example sub class of these attacks.

A key mechanism to limit the capabilities of these attacks is the SOP (Same Origin Policy), that restricts the type of scripts that one site can execute on the browser, in the context of a different site. Recently, attacks with the same general principle have been presented, to cause the victim's browser to access IoT devices and other resources on the victim's private network, behind a NAT.

We review the previously known attack by Acar et al. on an internal network, that uses DNS Rebinding to circumvent the SOP through sophisticated mapping of DNS entries to IoT devices in the internal network. This attack lets the attacker gain access to IoT devices, only by tricking the victim to surf to a malicious website.

Following this attack we present three additional attacks, each circumventing SOP in a different way. The first one is similar in its flow to the Acar at el. attack, but unlike it, instead of gaining access to IoT devices, we utilize DNS Rebinding and UPnP (Universal Plug & Play protocol) to directly attack the home router and gain access to the internal network. Thus demonstrating that home routers, which are considered also to be a kind of IoT devices, are the weakest point of the internal network.

75% of our tested home routers, that are supplied by ISPs in our region, are vulnerable to this attack. Next we demonstrate two more attacks on devices behind a NAT which do not use DNS rebinding. The first targets IoT devices and achieves same results as the attack of Acar et al., while the second bypasses previously known application security mitigations to attack home routers.

To circumvent these and similar attacks on devices behind a NAT we present Internal Network Policy (INP), a new natural complementary extension of the browsers security policy, SOP. INP prevents all attack types presented, and enhances the protection and security of private networks. INP can be used also for private networks without a NAT (e.g., IPv6), though it does not provide the same security isolation that a NAT provides. One of the major advantages of implementing defensive solutions in web browsers, is the frequency of browsers updates, in contrast to network devices as home routers or IoT devices.

INP stops attacks on devices behind a NAT that trick the victim to surf to a malicious website. There are other attack vectors for penetrating an internal network, such as getting physical access to the network, or exploiting a software vulnerability which communicates directly with the internet (such as an online chat or e-mail). These other attack vectors are more complicated, and are less common than attacks that use web browser's functionality. Notice, that these other attacks are supposedly stopped by IPS and firewall security devices.

After reviewing related work, we review the attack by Acar et al. and present three new attacks. Each attack highlights a different aspect of the root problem, all of which are solved by INP, the new security policy suggested here. We propose the INP defensive mechanism, explain how it works and why it protects from attacking behind a NAT, including a proof of concept implementation. In the end we show the experiments we have done to perform the attacks, check which devices are vulnerable and test the effectiveness of INP.

**Related Work**

Attacks on IoT behind NAT

In their paper "Web-based Attacks to Discover and Control Local IoT Devices" by Acar et al., an attack on IoT devices in home networks behind a NAT is demonstrated. The attack is based on the DNS Rebinding. Other attacks that uses DNS rebinding to attack private networks are presented at Dorsey's article.

Table 1 summarizes the differences between the attack of Acar et al. and the three new attacks that we present here. Two of the new attacks we present in section 2, achieve access to private networks without using DNS Rebinding.

Preventing attacks on the internal network

Many defensive approaches have been created to deal with attacking private networks behind a NAT. Some of them tried to solve DNS Rebinding, by creating specific rules in the home routers. Most of the given approaches, were difficult to implement and integrate with network devices. Moreover, many other application security mitigation concepts were invented, but vulnerabilities are still found and exploited, especially in IoT devices.

Johns et al. present extended Same Origin Policy "eSOP" . Johns et al. analyze the weaknesses of SOP, and point DNS Rebinding as the main problem which should be solved. eSOP was invented to combat DNS Rebinding, by using new headers which should be supplied by web servers in HTTP responses. In our paper, we analyze the problem of attacking internal network behind a NAT differently, and expose a different new root problem. A solution that is based only on DNS rebinding cannot prevent our second and third attacks, and eSOP does not stop our first attack either. The INP presented here copes with DNS Rebinding, but also with other attacks, including the new one presented in this paper. INP is secure by default, which means that if a vendor's device is not compatible with it, the external access to the device is prevented. Table 1 provides in addition a comparison of eSOP and SOP relative to these attacks.

Notice that our first attack, bypasses eSOP, even though the attack uses DNS rebinding. Although eSOP prevents an attack script from reading HTTP responses from other websites, our first attack needs nothing but sending a UPnP command in an HTTP request to the home router, the response is ignored.

Another IoT defensive approach is the concept of MUD (Manufacturer Usage Description), which lets IoT vendors provide a type of white-list specifying their device legitimate network behavior which can then be monitored and enforced by appropriate security devices. However, MUD relies on various IoT vendors to provide the MUD file. Moreover, the attacks shown in this paper, are not prevented by MUD, as they exploit the local HTTP servers of the IoT devices, which should legitimately be used in the LAN.

While implementing INP on Chromium we have noticed that in 2017 Mike West offered modifications to Chromium that head in the same direction as INP, preventing external resources from accessing internal network devices using the browser as a stepping stone. However, the INP presented here suggests a full solution addressing a wider range of issues not considered by the more ad hoc modifications of , including (i) dynamic and automatic internal IP classification, dealing with cases where the private IP addresses are not only according to RFC 1918. This is necessary in bigger organizations, such as large companies or universities, that do not always use RFC 1918 addresses for their internal networks. Furthermore, with the growing support for IPv6 many use cases for RFC 1918 addresses cease to exist, (ii) INP as presented here provides configurable rules that network administrators can manually add internal IP addresses on top of the default that should include the RFC 1918 ranges, and (iii) our INP preflight request provides details about the IP address of the request initiator thus enabling vendors of network devices (IoT, home routers, etc.) to allow cross site references if they come from specific external IP addresses such as the IoT vendor site. Moreover, the offered modifications by West do not prevent attacks which target internal devices by domain name instead of their IP, such as DNS Rebinding. In our implementation, we securely handle this cases.

**Attacks**

Four attacks that execute a script (originated in the attacker's website) on the web browser in order to access a device in the internal network are reviewed here. First we provide a basic background for SOP (Same Origin Policy). Then, we review the attack of Acar et al., and continue to present three new attacks.

Same Origin Policy

SOP (Same Origin Policy) is a standard security policy used by most browsers in order to prevent a malicious script on one page from obtaining access to sensitive data on another web page through that page's Document Object Model (DOM). An origin is defined as a combination of URI scheme, host name and port number. Under the policy, a web browser permits scripts contained in one web page to access data in a second web page, only if both web pages have the same origin. Static resources, such as images or frames, may be embedded cross-origin.

The security restrictions of SOP can be divided to two mechanisms:

1. Prevent accessing the response. If a script would send an HTTP request to a site from a different origin, SOP would prevent the script from reading the response, unless the other site allows it with corresponding HTTP headers.
2. Prevent sending complex requests. An HTTP request is considered complex, if it contains customized HTTP headers, or non trivial textual content type (otherwise the request is considered simple). If a script would send a complex HTTP request to a site from a different origin, SOP would block that request by default. Instead, SOP instructs the browser to send a preceding preflight request, which checks if the other site allows the original request to be sent. If the other site allows that with corresponding headers in the response to the preflight request, then the browser would send the original request.

Background: Acar et al. attack on IoTs

The attacker’s goal in the attack presented by Acar et al. is to access local devices in the victim's network in order to extract information from or send commands to the devices. The script which instructs the browser to make that access is placed in the attacker's evil.com website. First, the attacker tricks the premise user to visit its website using the script with the bait. A direct request to the IoT device points to a different origin (device IP address is different than evil.com), so the script would not be able to access the response. In order to perform the request on the local IoT device and overcome the SOP, DNS Rebinding is used, switching the IP of evil.com to the IP of the desired internal device.

The following is a step by step detail of the attack (see Figure 2):

1. The victim surfs to the attacker's website evil.com.
	1. The browser tries to resolve the IP address of evil.com, since the attacker's controlled response points to its external web server with a low TTL value (e.g. one second).
	2. The victim sends an HTTP GET request to evil.com, which responds with the attacker's malicious script.
2. The malicious script repeatedly sends GET requests to <http://evil.com/evil-test>, until the evil.com entry in the victim DNS cache is evicted (due to expired TTL). The request keeps going to evil.com and receives 200 OKs in response.
3. Attacker changes evil.com DNS record. When the entry's TTL expires, DNS rebinding by the attacker replaces the evil.com entry to point to the internal IP address of the desired device.
4. DNS Rebinding is completed. The victim tries again to resolve evil.com, which is now resolved to the IP address of the desired internal device.
5. Local device is accessed by the attacker. The attack script can now send HTTP requests directly to the HTTP server on the device and read responses, allowing the attacker to extract information from or send commands to the device.

In their paper, Acar et al. use an internal scan before the attack in order to detect specific IoT devices. This is because their attack targets IoT vendors, who they have previously researched.

Attack I: Attacking home routers with DNS Rebinding & UPnP

In this attack, the goal is to open a port in the NAT on the gateway. The instruction to open the port is placed in a script that is located in the attacker's evil.com web site. Most of the attack steps are similar to the previous ones presented by Acar et al. Nevertheless, this attack focuses directly on the home router, the single critical point of failure in the network. It lets the attacker access practically any device in the network without any prior scan. The attack is effective on 75% of the routers that are supplied by ISPs in our region, which we have verified.

Together with DNS rebinding, this attack takes advantage of the Universal Plug & Play (UPnP) protocol, used by most network devices, including home routers. A router's UPnP server supports HTTP requests which contain commands, such as adding a port forwarding rule or changing the DNS server.

The full attack flow can be viewed in Figure 2.

1. The victim surfs to the attacker's controlled website, evil.com.
	1. The same as in the Acar et al. attack above.
	2. The same as in the Acar et al. attack above.
2. Extracting the victim's gateway internal IP address. The WebRTC extension allows JavaScript to query the local client IP address. Acquiring the victim's internal IP address lets the attacker predict with a good probability of success the internal IP address of the gateway. Most of the networks share the same standards.
3. The malicious script repeatedly sends UPnP AddPortMapping requests to evil.com.
	1. Unlike the requests sent in the Acar et al. attack, UPnP HTTP requests are used here:
* Destination port may be different than 80 - which means the UPnP request is considered cross origin.
* The content type is XML - which means the UPnP request is considered complex (as explained in Subsection 2).

Therefore, preflight requests are triggered and sent to evil.com, which responds with corresponding "allow" headers.

Because the attacker lacks any prior knowledge about the victim's router, the attack script sends the UPnP command to evil.com to all the known UPnP ports in parallel.

* 1. The victim's browser now repeatedly sends UPnP AddPortMapping requests, as instructed in the script, to the web server at evil.com, which ignores the content and replies with HTTP 200 OK.
1. Attacker changes the evil.com DNS record. It’s the same as in the Acar et al. attack above, but it replaces the evil.com entry to point to the internal IP address of the router.
2. DNS Rebinding is completed. It’s the same as in the Acar et al. attack above. The victim resolves evil.com again, which is now resolved to the internal IP address of the gateway.
3. UPnP AddPortMapping request is sent to the gateway. The script in the victim's browser is still executed, and the victim sends the UPnP requests to evil.com – which is now the victim's home router. The attack goal is now achieved, a port forwarding rule is added, and the attacker can directly access the home network.

Except for the third step, most of the flow is very similar to the flow of Acar et al. In comparison to the attack by Acar et al., an initial scan of the internal network is not necessary.

The ability to add port forwarding rules to the home router is actually the significant step in taking over the victim's network, as local devices can be accessed directly. By using a Remote Code Execution (RCE) vulnerability or simple credentials, which are still common in many home devices, the attacker can gain control of the entire internal network.

In Section 6, we show that 75% of the examined routers supplied by the ISPs in our region are vulnerable to our attack. In Appendix A, we show how we are able to attack routers whose UPnP TCP port is randomized and dynamically changes. Routers whose UPnP URLs contain a long, random identifier, such as UUID, are not vulnerable, as the attacker cannot predict the URL to target.

Attack II: Attacking internal devices without DNS Rebinding

The second attack is based on the observation that SOP allows a site from one origin to send simple HTTP requests to a site from a different origin with no preceding preflight request (in contrast to complex origin requests). That means that if an IoT device can be controlled only by receiving a simple HTTP request, an attack script can exploit it, without DNS Rebinding. The request may utilize the device API, or in some cases exploit a vulnerability, such as BOF (Buffer Overflow), in order to execute code on the device.

Therefore, this attack demonstrates that a holistic solution to attacks behind a NAT, cannot be solved, by just preventing DNS rebinding.

In Figure 2, we demonstrate such an attack flow.

1. The victim is tricked to surf to the attacker's controlled website, evil.com - the browser retrieves the malicious script.
2. Through the malicious script, the attacker first scans the local network, to discover vulnerable devices (technical details available in Appendix A). Let us assume a vulnerable device is found with internal IP address 192.168.1.8.
3. The script sends a simple HTTP request to the device at 192.168.1.8, and completes the attack.

This attack may let an attacker, remotely control IoT such as multimedia devices (smart TVs, AV receivers) by their web management API, or even send crafted HTTP requests to exploit IoT vulnerabilities such as BOF (Buffer Overflow), which would lead to remote code execution on the device. In our experiments, we provide few examples of vulnerable IoT devices (see section 6).

Attack III: Attacking home routers through static HTML elements

This attack lets the attacker control the victim's home router, by accessing the router's web management interface through the victim's browser. Beside the familiar requirement of making the victim surf to the attacker's malicious evil.com, the attack also requires a geographical proximity to the victim. This prerequisite will be explained later in this subsection.

This attack makes the following contributions:

* It makes the victim's browser access the home router by using static HTML elements, instead of dynamic scripts. Therefore, SOP and other protection mechanisms such as the NoScript Firefox extension, are not effective since they are applied only to dynamic script elements. That proves that attacks behind a NAT could be performed in different ways than executing a script on the victim's browser.
* It enables the attacker to send any type of HTTP request (including complex ones) to the victim's router without DNS Rebinding. In comparison to the previous attacks, attack I sent UPnP requests, which are complex, but DNS Rebinding was required. Attack II didn't use DNS Rebinding, however it enabled the attacker to send only simple HTTP requests to the victim's devices in the internal network.

At high level, this attack exploits a vulnerability in the AP (Access Point) List feature of home routers.

Routers web interfaces which include this feature, expose a web page with an HTML table, that displays Wifi networks that are accessible to the router. One of the table's columns often present the networks' names (known as SSID), which are derived from networks' routers. With sufficient geographical proximity to the victim, an attacker which sets a Wifi hotspot can control one of the entries in the victim's router AP List table. A basic Wifi hotspot (e.g. created by a smartphone), would be accessible to a router from a maximum distance of less than hundred meters. However, depending on the attacker's antenna and transmitting abilities, it can be accessible from hundreds of kilometers.

If the attacker sets its network name to be an HTML script tag, such as <script src="http://evil.com/evil.js"/>, and the victim's router AP List web page does not encode or handle networks' names properly, then the attacker's network name would be interpreted as native HTML.

If the victim surfs to evil.com while the malicious hotspot is accessible to the victim's router, the only thing left for the attacker to complete the attack is to redirect the victim from evil.com to the router's AP List page. Then the attacker's controlled script (located at http://evil.com/evil.js) is executed on the victim's browser. The most significant consequence is that the attacker's script is executed in the context of the victim's router AP List page, which means that it can freely access the router's web interface without any restriction by SOP.

This vulnerability belongs to a large class of vulnerabilities, named XSS (Cross Site Scripting). Previous similar XSS vulnerabilities in home routers have been shown, however they required the victim to surf directly to the router vulnerable page (instead of evil.com), which dramatically decrease the probability of success.

Following is a step by step detail of the attack (see Figure 3):

1. The attacker sets up an access point, with network name: <script src="http://evil.com/evil.js"/>. The access point should be reachable by the victim's router.
2. The victim surfs to the attacker's controlled website, evil.com - the browser retrieves the malicious HTML page.
3. The malicious HTML identifies the victim's router, taking advantage of HTML image element scan technique (detailed in Appendix A).
4. The attacker's HTML makes a login request (using HTML form), to the home router. We assume the credentials are default ones. This makes the active session in the router, to be the victim's one.
5. The attacker's HTML redirects the victim (by setting the HTML document location), to the AP List panel page.
6. The XSS is invoked. The attacker's access point's network name, is now parsed as HTML, and a prepared script (at http://evil.com/evil.js) is ready to be executed, in the context of the router itself.

Later in our experiments (section 6), we show that 41% of the tested routers which are supplied by ISPs in our country, are vulnerable to our attack.

**INP**

Motivation

The key element in all the presented attacks on internal networks behind a NAT is a request that is initiated by an external server, and sent to an internal network resource. As SOP deals with cross origin the discussed attacks use cross network, access an internal network resource in the context of an external web-page.

We define the following terms, which we later use in this section.

An *internal IP address*, regarding to a network device, is an IP address which matches one of the followings:

1. the network subnet of the device.
2. the address spaces in an XML configuration file prepared by the network admin (will be discussed in subsection 4).
3. For an IPv4 address:
	1. the loopback space (127.0.0.0/8) defined in section 3.2.1.3 of RFC 1122.
	2. the link-local space 169.254.0.0/16 defined in RFC 3927.
	3. the address space defined in Section 3 of RFC 1918.
4. For an IPv6 address:
	1. the Local Address prefix (fc00::/7) defined in Section 3 of RFC 4193.
	2. link-local prefix (fe80::/10) defined in section 2.5.6 of RFC 4291.

An *external IP address*, regarding to a network device, is an IP address which is not an internal address regarding to that device.

A *cross-network HTTP request*, is an HTTP request which was initiated by a resource with an external IP address, and accesses a resource with an internal IP address.

From the attacker point of view, the previous shown attacks are different from each other and do not share a common component (as can be seen in Table 1). Nevertheless, from the victim's web browser's perspective, all the attacks share a similar flow which we present in Figure 3:

* The attacker's evil.com website is hosted in the internet, with an external IP address, regarding to the victim. The attacker tricks the victim to surf to that malicious website.
* The victim's browser receives the response from evil.com. No matter which technique the attacker uses, either dynamic script with DNS Rebinding as in Attack I, or static HTML content without DNS Rebinding as in Attack III, the response refers to a resource with an internal IP address.
* The victim's browser send the the attacker's request, which was embedded in the response from evil.com, to the wanted internal network resource. As we defined earlier, this request is a cross-network request.
* The device in the victim's internal network receives the attacker's cross-network request, thus the attack is completed successfully.

INP Policy

In this section we explain the details of our suggested policy, INP.

The basic idea of INP is similar to SOP. At a high level, while SOP applies restrictions between web origins, INP applies restrictions between external and internal network resources. In the same way preflight requests are used by SOP to treat complex requests (as explained in subsection 2), INP treats cross-network requests.

Figure 4, describes the process of cross-network requests handling by the web browser. Whenever the browser notices a cross-network request, it first checks whether a previous positive preflight result, which fits the cross-network request parameters (source IP, method, etc.), is already cached. Unless a positive result is cached, the browser sends a preflight request (its headers will be presented later) to the destination of the cross-network request. According to the preflight response, the browser either blocks or send the cross-network request.

As SOP tracks a request's origin, INP tracks its initiator's IP address. The IP address space classification will be described later in the implementation.

Table 3 presents the HTTP headers used by INP both in the preflight request and response. The request headers provide the internal resource information about which external entity is trying to access it (web origin and IP address) and how it plans to do it (method and customized headers of the cross-network request). The response headers indicate the browser whether the internal resource allows the cross-network request or not. An additional header in the response is X-INP-Max-Age, which gives the number of seconds response to the preflight request can be cached for without sending another preflight request.

Security Evaluation

INP is a secure by default. Due to its definition, no cross-network HTTP requests (regarding the private network), are sent without an explicit approval. Only if an internal network resource explicitly allows a resource with an external IP address to access it with a cross-network request, then the request would be sent by the browser.

Therefore, INP prevents by design attacks behind a NAT which use the browser as a stepping stone, including those we have presented in this paper.

We demonstrate how INP prevents the previously shown attacks:

* Background attack - Acar et al. - after DNS Rebinding occurs, the HTTP request, which was initiated by the script from evil.com with an external address, accesses an IoT device with an internal address. Therefore this request is a cross-network request. The browser would then send an INP preflight request to the device, which would probably not accept it. For that reason, the attacker's cross-network request would not be sent.
* Attack I - INP acts here just as the same as the case of Acar et al. attack. Instead of sending the INP preflight request to an IoT device, it would be sent to the home router.
* Attack II - the attack script from evil.com, triggers a request to an internal device. This is of course a cross-network request, which causes INP to send a preflight request to the device.
* Attack III - in this case, we show that INP handles request which were not only initiated by a dynamic script, but also from static HTML elements. INP would prevent the two steps which are required for the attack:
	+ Step of HTML form - which sends the login request to the home router which is classified as a cross-network request.
	+ Redirection - the attacker attempts to redirect the victim's browser from evil.com to the home router. However, this redirection is also a cross-network request, as it is originated from evil.com with an external address and its target is the home router whose address is an internal one.

Functional Evaluation

First of all, INP provides full backward compatibility if it is not implemented:

* If the browser does not support INP - all the HTTP traffic is handled classically, controlled by mechanisms as SOP, CORS, etc.
* If an internal network device's web server does not support INP - as explained in the previous subsection 2, the cross-network request would not be sent to the internal device, protecting it from a potential attack.

One of the major advantages of INP, is the fact that it is implemented in the web browser. This is critical as web browsers are updated in a higher frequently than devices such as routers supplied by ISPs. That lets us believe that INP would be much more comfortable for integration, than other previous solutions, such as those which are supposed to be implemented in routers.

A major concern during designing INP was to avoid breaking any existing functionality in internal network resources, such as IoT devices. For example, INP allows by default an internal resource, to send HTTP requests through the user's browser to another resources in the internal network. There are systems, such as smart homes, which use this internal access as part of their functionality. As will be described in our experiments, we checked if there is any access by design, from an external entity (e.g. a cloud server) to an internal network IoT device, through the web browser. We did not find one at all. Additionally, if a user wishes to surf directly to an internal network resource (such as an IoT web interface), it is recognized and allowed by INP.

INP and SOP are naturally integrated together. In cases where a cross-network request is also a cross-origin complex one (this term was discussed in subsection 1), then the browser would send a preflight request with both INP and SOP relevant headers. When the preflight response is received, both INP and SOP validations would occur.

Practical Implementation

In order to validate the feasibility, security and functionality properties of the INP, we implemented a proof of concept of it for the Chromium web browser. In this subsection, we deal with some of the practical challenges of INP.

Address space classification: To learn the internal network address space, beside using static known addresses (such as those in RFC 1918), the browser can simply use the API of the machine it is running on (Windows, Linux, etc.) to get the network interfaces, IP addresses and subnet masks (both IPv4 and IPv6). The main advantage of that address space discovery is the ability to achieve the internal addresses precisely and dynamically, without relying only on internal RFC 1918 IP ranges. For compatibility with larger organizational networks, which contain more than one address space, INP offers the network administrators the option to manually configure the organizational internal network spaces. This can easily be implemented with an XML configuration file, which spreads in the network through management protocols such as GPO. An example of an XML configuration file is shown in Figure 2 in Appendix B.

Tracking the request's initiator's IP address: To determine if an HTTP request is a cross-network one or not, the browser should track the IP address of the request's initiator. In the same way as the tracking of every request's origin, by SOP, is implemented in web browsers, the corresponding tracking of IP addresses should be implemented.

In our proof of concept in Chromium, we relied on a previous work to track whether an HTTP request is cross-network or not. We overrode the code which classify IP address spaces and added the INP functionality in preflight requests.

Additionally, we added code which handles cross-network requests that target DNS names. In the current Chromium implementation, requests are determined cross-network before the DNS resolution takes place by the browser. Therefore, a cross-network request whose destination host is a DNS name of an internal network device, would not be considered cross-network, as the browser still does not know the IP address of the request's destination. Thus attacks as DNS Rebinding would not be prevented by the current modifications offered in Chromium. In order to solve this case, we added a prior DNS resolution.

More technical details about our proof of concept implementation can be viewed in Appendix B.

**Experiments**

In this section, we review the experiments we have done during our research. These include the setup of our three presented attacks, along with testing of our INP proof of concept.

Attack I - Attacking home routers with DNS Rebinding & UPnP

In order to simulate the attack execution, we used an Amazon Linux server to host the attacker's malicious website (referred to earlier as evil.com). For the DNS rebinding simulation, we set the victim's DNS server to be the Amazon server. The routers we chose to test the attack on are the most popular among the largest ISPs in our region, as verified by 200 students. We tested the whole cycle with each vendor, from surfing to the attacker's website to attempting to send the UPnP commands to the router.

Most of the routers which enable UPnP in the LAN are vulnerable to our attack. Routers that are not vulnerable usually include a unique identifier in the UPnP URL (such us UUID). As the attacker does not have direct access to the router's LAN, the identifier cannot be learned; thus, the UPnP server cannot be accessed.

By sending an SSDP M-SEARCH request to a router, its UPnP server (if available) would respond with the server URL. This provides a researcher enough information, whether the router is vulnerable or not.

Table 1, presents the router distribution and the results of the experiments. More specific details about the vendors and the exact exploitation (such as the UPnP URL) can be viewed in Appendix A. As seen in the table, 75% of the tested routers are vulnerable to the attack.

Attack II - Attacking internal devices without DNS Rebinding

The setup for this attack is quite simple, as we only use an Amazon Linux server to host our malicious website. We examined two IoT devices at home and created a proper attack script to control them.

Yamaha Network AV Receiver – The Yamaha RX-V683 Network AV Receiver exposes a web management interface on port 80. An attacker can send specific commands to the URL: "/YamahaRemoteControl/ctrl" to take control of the device. The commands are sent in HTTP POST requests without any authentication or customized headers needed. The body of the request is an XML (although no Content-Type HTTP header is necessary). Beside the web management interface, which listens on port 80, we discovered that the Yamaha RX-V683 Network AV Receiver exposes an interface which listens on port 49154.

Therefore, after scanning the internal network for a device which listens on this unique port, we sent the relevant HTTP request to control the device.

Sony Smart TV - An attacker can send commands (HTTP POST request) without authentication directly to the HTTP web server of a Sony Smart TV with the URL: "/sony/IRCC?." The same as in the case of Yamaha, the command itself is an XML which appears in the body of the HTTP request. Sony supports many different commands, such as Power Off, Volume Control, and Display Control, etc.

Attack III - Attacking home routers through static HTML elements

Beside the website we host at Amazon, which acts as evil.com, in this attack, we also used our smartphone to set up a mobile hotspot. We tested the same routers as in Attack I. A simple experiment of creating a hotspot (with an HTML script tag as its name) and surfing to the router's AP List page can indicate whether the router is vulnerable to XSS.

We used the network name <script>alert("HACKED")</script>, which popped up a message box in the victim's browser (see Figure 3) when surfing to the AP List page. We tested the full attack flow rather than this vulnerability alone. 41% of the tested routers were vulnerable to the attack. The results are presented in Table 1.

Responsible vulnerability disclosure

We reported vulnerabilities discovered throughout our research to the respective router (D-Link, TP-Link, VTech, Sagemcom, ADB) and IoT (Yamaha, Sony) vendors. Some of their security response teams are in contact with us, working together to fix the vulnerabilities.

Testing INP

As described in the INP implementation (Subsection 3), to provide a proof of concept, we developed our version of Chromium, which applies INP. We tested the effectiveness of INP against our attacks in our compiled version of Chromium on a Windows 10 machine.

Our INP implementation successfully prevented all of our attacks by sending a preflight request to the internal network target instead of the attacker's malicious cross-site request.

We also examined the performance difference between the INP version of Chromium and the original one.

As discussed earlier, the implementation manifests itself mostly as parsing and extraction of the HTTP headers and string comparison in the address space classification. The comparison is executed for any request, and the header parsing is only for INP preflight requests.

In order to handle requests whose destination host is not an IP address but a DNS name, we added a prior DNS resolution step, as explained in Subsection 3.

In the beginning, our DNS resolution occurred at every processing by Chromium of any HTTP request. This dramatically lowered the browser's performance. Websites which were usually loaded by Chromium after a couple of seconds, were loaded by our browser version after a minute.

Then, we made a significant improvement by removing most of the times where the DNS resolution took place, keeping only the required time for correctly determining cross-network requests (technical details can be viewed in Appendix B).

We had practically no noticeable overhead when accessing a web application in our tests.

**Conclusion**

In this paper, we have spotted the key vulnerability in most of the attacks behind a NAT, which use the web browser as a stepping stone. We realized that no matter which attacks are used, the browser would eventually be instructed by an external entity to access an internal network resource. Many studies have been conducted over the years, most of them trying to prevent specific issues, while the root problem still exists. We have shown three attacks, each of which presents a different perspective of the problem. We then presented an Internal Network Policy (INP), which deals with and solves the main problem and aims to protect users and organizations from being attacked and penetrated.