Stopping DNS Rebinding Attacks in the Browser

Mohammadreza Hazhirpasand, Arash Ale Ebrahim and Oscar Nierstrasz
University of Bern, Switzerland

Keywords: DNS Rebinding, Browser Security, Web Security.

Abstract: DNS rebinding attacks circumvent the same-origin policy of browsers and severely jeopardize user privacy. Although recent studies have shown that DNS rebinding attacks pose severe security threats to users, up to now little effort has been spent to assess the effectiveness of known solutions to prevent such attacks. We have carried out such a study to assess the protective measures proposed in prior studies. We found that none of the recommended techniques can entirely halt this attack due to various factors, *e.g.,* network layer encryption renders packet inspection infeasible. Examining the previous problematic factors, we realize that a protective measure must be implemented at the browser-level. Therefore, we propose a defensive measure, a browser plug-in called Fail-rebind, that can detect, inform, and protect users in the event of an attack. Afterwards, we discuss the merits and limitations of our method compared to prior methods. Our findings suggest that Fail-rebind does not necessitate expert knowledge, works on different OSes and smart devices, and is independent of networks and location.

1 INTRODUCTION

In a DNS rebinding attack, an attacker uses a custom DNS server to spoof the IP address of the victim, and thus obtain read access to the victim's server. DNS rebinding attacks have been around for more than 15 years (Dean et al., 1996). Being cost effective and relatively quick to perform, this attack is capable of jeopardizing both the victim and the intranet to which he or she is connected.

To orchestrate a DNS rebinding attack, the attacker runs a malicious website, *e.g.,* attacker.xyz, with a custom DNS server. Once the victim loads *attacker.xyz*, the website establishes a request in the background to its server. The malicious server returns the private IP address of the victim as a response. Consequently, SOP in the victim's browser is tricked into believing that the two IP addresses belong to *attacker.xyz* and grants read access to the malicious website.

Web browsers employ same-origin policy (SOP) to provide isolation for distinct origins (scheme, host, and port of a URL) to protect users from different types of attacks, *e.g.,* cross-site request forgery, or clickjacking (Lalia and Moustafa, 2019). However, the DNS rebinding attack subverts the SOP and turns a browser into an open proxy to access the victim's resources and other nodes inside the victim's network. In practice, an attacker executes DNS rebinding to bypass firewalls, gain access to either internal machines in the victim's private network or the victim's resources, fetch sensitive resources, and compromise vulnerable internal machines.

The DNS rebinding attack can have grave consequences. For instance, a study showed that popular home routers such as Linksys, Thompson, Belkin, Dell, and Asus are vulnerable to DNS rebinding, threating the privacy of millions of users (Heffner, 2010). Tatang *et al.* analyzed four Internet of Things (IoT) devices and found three of them are vulnerable to DNS rebinding attacks (Tatang et al., 2019). They emphasized that most smart home devices heavily rely on the countermeasures implemented in the router to detect and block such attacks. Similarly, Dorsey in a whitepaper explained that widely-used smart devices such as Google Home mini, RokuTV, Sonos speakers, and home thermostats are controllable by the attacker from outside of the victim's private network.[1]

There exist several measures to prevent DNS rebinding attacks. Modern browsers apply a technique called DNS pinning in order to cache the first resolved IP address of a website for a fixed period. Even though this technique discards the time to live (TTL)

[1] https://medium.com/@brannondorsey/attacking-private-networks-from-the-internet-with-dns-rebinding-ea7098a2d325

Hazhirpasand, M., Ale Ebrahim, A. and Nierstrasz, O.
Stopping DNS Rebinding Attacks in the Browser.
DOI: 10.5220/0010310705960603
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 596-603
ISBN: 978-989-758-491-6

of each DNS record, the browsers reset the cached DNS data in case the victim spends approximately more than a minute on the attacker's website. There exists the Domain Name System Security Extensions (DNSSEC) specification suite to prevent DNS spoofing attacks. For domains owners, DNSSEC provides a way to authenticate DNS entries in order to ensure the response has not been altered in transit by an adversary. Notwithstanding authentication, in a DNS rebinding scenario, the attacker is the owner of the domain name, *e.g.,* attacker.xyz, and hence, DNSSEC becomes ineffective. Firewalls, *e.g.,* iptables, at the entry point of networks can detect such attacks by defining particular rules. Despite firewall packet inspections, the victim can still be unprotected while using a virtual private network (VPN) that encrypts DNS queries. Lastly, none of the aforementioned techniques can keep a user completely safe from DNS rebinding attacks.

In this paper, we investigate and discuss the effectiveness of available countermeasures in protecting users against DNS rebinding attacks, and propose a preventive measure, called Fail-rebind, based on lessons learned from drawbacks of the prior preventive measures. To that end, we address the following research questions: *"How can the current preventive measures be circumvented, and what is the most robust way to detect and halt DNS rebinding attacks?"*

We argue that the preventive shield must be implemented in browsers rather than in the network or operating system level. To this end, we propose a Chrome and Firefox plug-in, called Fail-rebind, to detect, alert, and block the malicious website in the event of the attack. In our comparison with other methods, we observe that our proposed solution is highly reliable, practical in daily use, and can significantly increase safety against DNS rebinding attacks.

The remainder of this paper is structured as follows. In section 2, we explain various types of DNS rebinding attacks, current attack frameworks, and an example attack scenario. Then, in section 3 we discuss the pros and cons of existing countermeasures introduced in prior research. We propose and evaluate our preventive measure in section 4, and conclude the paper in section 5.

2 BACKGROUND

We first describe various types of DNS rebinding attacks as well as the available attack frameworks. We then explain how impactful the attack is in a given scenario, and why we need a robust and reliable prevention measure.

2.1 Attack Types

An attacker can take advantage of DNS rebinding attacks to achieve various goals. Depending on the goal, the impact of DNS rebinding attacks is classified into two main categories, *i.e.,* firewall circumvention, and IP hijacking.

2.1.1 Firewall Circumvention

Network administrators consider firewalls to be an effective defensive barrier against inbound attacks against internal machines. However, firewall circumvention techniques, *e.g.,* DNS rebinding attacks, enable the attacker to gain access to the victim's local resources and internal nodes of the victim's network. Attackers are keen on achieving the following objectives:

— *Data stealing.* Web servers in a private network might expose confidential documents to legitimate users inside the network. However, the attacker can steer DNS rebinding attacks in such a way as to read and download sensitive information from the victim's local resources or the nodes in the victim's private network.

— *Exploitation.* With due consideration of the unpatched victim's local or remote services in the victim's network, the attacker can perform reconnaissance and exploitation attacks against vulnerable nodes, *e.g.,* home routers or an internal machine with an unpatched WordPress website.

2.1.2 IP Hijacking

DNS rebinding threatens both the internal nodes in the victim's private network and machines on the internet. In the following, we explain two types of this attack threatening external nodes.

— *Click fraud.* The attacker can mount a click fraud attack in which users inadvertently click on an item leading them to browse a malicious target webpage. In practice, when *attacker.xyz* is loaded on the victim's browser, the attacker can reply to the second DNS query with an IP address of a target website, *e.g., clicktopay.xyz.* Conducting this attack on a large scale results in thousands of unique IP addresses visiting *clicktopay.xyz* to generate profits (Jackson et al., 2009).

— *Bypassing IP-based authentication.* Services can rely on IP-based authentication to impose restrictions. However, the attacker can simply defeat such measures by using DNS rebinding, as the communication originates from an authorized IP address (the victim's browser).

Unfortunately, in all the above scenarios, the target machine logs the address of victim machine for illegal activities.

We manually checked how many repositories are associated with DNS rebinding on GitHub. We searched for "DNS rebinding" and found 47 repositories. Of these, two repositories offered protection in the form of middleware using the Go language and the Express.js framework, and the rest of the repositories all introduced attack frameworks mainly written in JavaScript and Python.

Of the 45 attack frameworks, we examined Singularity, a recently popular and rich DNS rebinding attack framework, proposed by Doussot *et al.*[2] At the time of writing, this framework has received the highest number of stars and forks on GitHub compared to other DNS rebinding attack frameworks. Singularity offers numerous features such as a custom DNS server, IP/port scanning, auto exploitation, and DNS CNAME to evade firewall restriction and interactively browse the victim's internal network. Singularity's main contribution is to significantly improve the rebinding time in browsers by 20 times (from 60 seconds to 3 seconds) and introduce three circumvention methods for current protection measures. It is inevitable that such advances imply the ineffectiveness of the current prevention techniques in this domain.

2.2 Motivational Example

Our goal is to demonstrate the grave implication of DNS rebinding attacks on the privacy of users in a scenario (See Figure 1). We use a laptop, a Plant Watering System (PWS) smart device and a mobile phone as nodes in the network. The mobile phone has an application in which an HTTP web server is running to allow users to browse the phone's picture gallery. The PWS has a web interface where features such as auto watering on/off, soil status, and watering history are accessible. We use the iptables firewall to prevent external domains from resolving private IP addresses and loopback IP addresses. Our gateway to the internet is a router configured with OpenDNS. The OpenDNS service extends the DNS by introducing security features such as phishing protection, content filtering, and DNS rebinding prevention. The client on the laptop uses a secure VPN connection, which encapsulates and encrypts all network traffic, and sends the user's data through a tunnel to a server located on the internet. The VPN connection forces the DNS lookup procedure to be accomplished through the VPN server. On the internet, the

[2]https://github.com/nccgroup/singularity

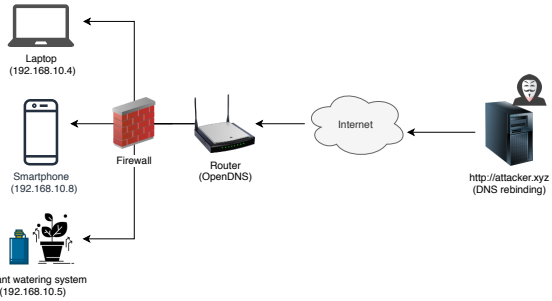

Figure 1: The network scenario of the motivational example.

attacker runs a web server with a domain name, *i.e.,* attacker.xyz, and a custom DNS server.

Attackers commonly find the private IP address of a victim with the help of the IP leakage in WebRTC and carry out IP or port scanning, *e.g.,* loading images or XMLHttpRequest, in order to find active nodes based on timing thresholds (Hazhirpasand and Ghafari, 2018). In this scenario, however, the victim employs a secure VPN connection wherein no private IP leakage occurs and WebRTC returns the real public IP address of the victim. As a result, the attacker must execute preliminary IP scanning to discover the victim's private IP range. Finding the private IP range can dramatically decrease the success rate of the attack provided that the network administrator sets an uncommon range of private IP addresses, *e.g.,* addresses in the IPv4 class A.

In our experiment, we scanned 400 common private IP address ranges used by popular routers or network administrators in 40 seconds. When the private range is found, the JavaScript program starts to scan the entire range (1-255), find active nodes, and send the private IP addresses of the active nodes back to the server. In this scenario, the attacker sets the IP address of the PWS on the server's custom DNS server. Afterward, the malicious page starts fetching the *http://attacker.xyz/* website every 500 milliseconds in the background. In our experiment, it took 71 seconds in the Firefox browser to fetch the new DNS information. This delay in fetching the new IP address was due to the existence of the DNS pinning feature in browsers. In addition, the same-origin policy did not complain as the domain name remained the same. The malicious website was then allowed to send POST and GET requests to the PWS web interface. For example, the attacker was able to send a GET request to turn off the auto watering system.

The failure in the OpenDNS service and the firewall in detecting the attack can be explained by the fact that the VPN connection encrypts all the traffic and that DNS server of the VPN server resolves the

attacker's website. Moreover, we observe that if the user spends nearly 2 minutes on the malicious web page, the existing countermeasure on the browsers will lose its efficacy.

3 EXISTING MEASURES

We now introduce and assess how the widely-known preventive techniques provide inadequate safety for users. We classified the proposed solutions into two broad categories, namely browser-based and server/network-based solutions. Table 2 provides an overview of existing preventive solutions and their drawbacks. In the following, we discuss each of them in detail.

3.1 Browser-based Solutions

Browsers implement the DNS pinning approach to block DNS rebinding attacks. DNS pinning protects both private and public nodes but for a short period. They cache the IP address of the hostname for a fixed period when it is loaded. However, browsers cannot keep such information for a long duration as web applications may need to switch to another server due to various reasons, *e.g.,* the denial of service attack or load balancing. Consequently, they update their cache after a short period, ranging from several seconds to minutes. Table 1 presents the average required time to conduct a successful DNS rebinding attack on different browsers. We performed an experiment for each browser ten times, and employed the FakeDNS server script.[3]

Despite the short period for DNS pinning, attackers used creative approaches to successfully circumvent DNS pinning in browsers. Dai *et al.* dissected DNS pinning in order to find a way to shorten the fixed period in browsers (Dai and Resig, 2013). They exploited DNS pinning by flooding the DNS cache table on the victim's browser with numerous invalid entries to persuade the browser to do a second DNS lookup. Their tool, called FireDrill, utilizes a WebSocket connection to enable the attacker to have an interactive session with the victim's browser. Jackson *et al.* in a study suggested a number of defenses for the socket access policies in Flash Player, Silverlight, and JVM (Jackson et al., 2009). Their solutions were adapted by Adobe, Microsoft, and Sun in order to provide a secure socket level access for plugins.

The pinning can be extended with heuristics to enhance its defense level. For instance, browsers can

Table 1: Average time of a successful DNS rebinding attack on different browsers.

Browser	Version	OS	Avg time
Google Chrome	81	Windows 10	2:05s
Chromium	80	Kali Linux	2:11s
Mozilla Firefox	75	Windows 10	1:15s
Mozilla Firefox	68	Kali Linux	1:43s

enforce a policy to allow hostnames to rebind to any IP addresses within a /24 network, *e.g.,* attacker.xyz with an IP address of 190.145.12.22 can rebind to any address with the range 190.145.12.X. Incorporating such IP-based origins into browsers might have negative consequences for real-time web applications whose JavaScript WebSocket or long polling runs for a long period and might switch to other servers due to a failover. Unfortunately, since many distributed DNS services, *e.g.,* Cloudflare, have servers across continents, it adversely affects the trade-off between availability and security. Furthermore, each browser currently uses a separate pin database to store DNS replies. Having a centralized pin database can reduce the collaborative effort among different browsers to build a comprehensive defense measure against DNS rebinding attacks.

A browser-level solution to DNS rebinding is the NoScript plug-in for the Firefox browser.[4] This plug-in is intended to shield users from DNS rebinding attacks against private nodes. Released in 2005, the plug-in protects users against web-based attacks such as XSS, CSRF, clickjacking, man-in-the-middle, and DNS rebinding attacks. The plug-in blocks any scripts including JavaScript, Java, Flash and Silverlight by default in Firefox. In our experiment, when we enabled JavaScript on the attacker's website, the plug-in did not detect any incoming DNS rebinding attacks. Consequently, we found this tool not suitable for day-to-day browsing as it completely blocks a website's JavaScript to protect users, which considerably reduces the functionality of the website.

Johns *et al.* extend SOP (named eSOP) for the Chromium Web browser to defend users from DNS rebinding attacks, and they describe how to utilize the HTML5 Offline Application Cache to carry out a reliable DNS Rebinding attack (Johns et al., 2013). In their proposal, a dedicated HTTP server response (X-Server-Origin) is introduced to transfer the server origin property in the form of a comma-separated list. When the attacker sends a request to the victim machine, eSOP checks the X-Server-Origin header of the victim and marks the communication as invalid due to inequality of the attacker's domain name and the victim's server origin, even though the classic (protocol/domain/port) SOP is satisfied.

[3]https://github.com/pathes/fakedns

[4]https://noscript.net/

Table 2: Preventive measures for DNS rebinding attacks and their drawbacks.

Approach	Type	Drawbacks
DNS pinning	Browser	Limited pinning time, can be shorten by DNS flooding
DNS pinning - Extended IP check	Browser	Not applicable in real world scenarios and impose many limitations
A centralized pin database	Browser OS	Necessitates of collaboration between OS and browser vendors
NoScript plug-in	Browser	Blocks all javascript contents in a website
eSOP	Browser	Requires a server header, a custom browser, cannot detect IP/port scanning
INP	Browser	Requires implementation in browsers, and no detection for IP/port scanning
Brahmasani et al	OS - Network	Encrypted packets cannot be read - can be only used by desktop computers
dnswall	Network	Encrypted packets cannot be read
Host header	Server	Requires manual work, knowledge of hostname, erroneous implementation
Security Proxy	Network	Lack of implementation, and high overhead
SSL	Server	Requires SSL on all nodes, problematic for programs with an invalid SSL
OpenDNS / Dnsmasq	Network	Encrypted packets cannot be read

Similarly, Afek *et al.* present a prevention mechanism, called Internal Network Policy (INP), to stop three common attacks including DNS rebinding attacks (Afek et al., 2019). In general, INP acts as a new browser security policy by preventing access to internal resources from external entities. They emphasize that INP is complementary to SOP, as SOP only considers cross-origin access and overlooks IP addresses. They implement a proof-of-concept (PoC) of INP in the Chromium web browser. They extract IP addresses after any required DNS resolution, just before the request is about to be made. In case the destination IP address has not yet been resolved, there is a small chance that the DNS rebinding attack can take place. Their comparison with eSOP reveals that INP delivers better security as eSOP is incapable of protecting home routers or machines that are vulnerable to remote command execution.

3.2 Network-based Solution

Brahmasani *et al.* propose a two-level defensive solution analyzing all incoming packets on DNS port (*i.e.,* 53) (Brahmasani and Sivasankar, 2013). First, they compare the hostname associated with the canonical name of each reverse DNS lookup with the original domain name, and in the second level they check the HTTP response content of each IP address returned by the DNS response. However, the solution suffers from the same drawback associated with the encrypted packets when the victim uses a VPN connection.

Bortz *et al.* propose a remedy to the DNS rebinding problem. They introduce and implement an open-source DNS resolver, called dnswall, to prevent external hostnames from resolving to an internal IP address. The implementation is included in FreeBSD.[5] Subsequently, similar means of defense are implemented in other open-source firewall projects such as Dnsmasq, OpenWrt, DD-WRT (Fainelli, 2008).[6] Software firewalls can also block DNS resolutions to the loopback interface but this defense is not adequate as services bind to other network interfaces. Both defensive measures can be inadvertently circumvented by the victim when using a VPN connection. Furthermore, restricting access to private IP ranges, as defined in RFC 1918 (Rekhter et al., 1996), is not necessarily a comprehensive solution. This is due to the fact that large companies or universities do not always use RFC 1918 addresses for their internal nodes. Craig Heffner explained that the attacker can use DNS rebinding to gain access to private nodes in a local network, if such machines are configured both on a private and a public IP address (Heffner, 2010).

An semi-robust obstacle to DNS rebinding attacks is to compare the HTTP host header. In DNS rebinding, the attacker cannot spoof a fake host header and accordingly, the victim's web application/server is able to check whether the host header is consistent with the address of the requested website. This technique requires developers or server managers to manually add the security check to their either web application or web server (See Listing 1). Notably, security checks can be error-prone or implemented incompletely on servers with many hostnames, which allows the attacker to access a subset of the victim's data. This technique cannot be employed on servers or applications with no prior knowledge of their hostname. Lastly, this technique provides partial robustness and safety as the attacker can still trigger request and receive the response from the target server. This may result in some valuable data leaks regarding the target defensive strategies.

[5]https://svnweb.freebsd.org/ports/head/dns/dnswall/

[6]https://dd-wrt.com/

Listing 1: The host header check in the entire application.

```
if ( $_SERVER['HTTP_HOST'] == "localhost") {
  // Application code runs
} else {
  echo "Something is wrong";
}
```

Pandiaraja *et al.* propose a defensive system, called Security proxy, through the use of the Advanced Encryption Standard encryption algorithm and a hash function (Pandiaraja and Parasuraman, 2015). The Security proxy system provides a secure environment for DNS queries to be securely communicated. However, the proposed system is not evaluated from the security, performance or feasibility point of view.

A website can protect itself from being targeted by DNS rebinding attacks by having an SSL certificate on the server side. When an attacker attempts to bind *attacker.xyz* to the IP address of a website secured with SSL, browsers consider the certificate invalid due to the mismatched domain name (attacker.xyz). As a result, once the new IP is set, the background JavaScript code cannot fetch the source code of the target website and receives HTTP 302 Found redirect status. In our analysis, browsers return a warning page indicating the SSL certificate is not trusted, and fortunately, the attacker cannot dismiss this error message programmatically. We also realized that having a signed or self-signed certificate has the same preventive effect on halting the DNS rebinding attack. Importantly, SSL can be ineffective when the web server does not force SSL on all incoming connections. A recent study has shown that a large number of websites do not enforce the usage of digital signatures (Van Goethem et al., 2014). This leaves the opportunity open for attackers to conduct DNS rebinding attacks aimed at public nodes. Our finding confirms that using either the .htaccess file or HTTP Strict Transport Security (HSTS) is mandatory. However, misconfiguring the HSTS response header in a way that the web server only returns the header without forcing HTTP to HTTPS connections would render it ineffective. We examined commonly used web servers or server software with default configurations to observe whether any software forces the use of HTTPs (See Table 3). The results offer compelling evidence that no evaluated software forces users to use a digital certificate. Furthermore, users might consider forcing HTTPS over HTTP unnecessary in a private network.

In Tatang *et al.* experiment, the Teltonika RUT500 router is capable of detecting DNS rebinding attacks owing to the activation of the DNS rebinding protection in Dnsmasq (Tatang et al., 2019). Nevertheless, they noticed that the user's localhost is still vulnerable. Similar to OpenDNS, Dnsmasq is unable to detect the attack when the victim uses a VPN connection.

Table 3: Forcing to use SSL in different web servers and server software.

Web server	Version	Operation System	Forced SSL
XAMPP	7.2.33	MacOS	No
XAMPP	7.2.33	Windows 10	No
WampServer	3.2.0	Windows 10	No
IIS	10	Windows 10	No
XAMPP	7.2.3	Kali Linux	No
LAMP	7.3.2	Kali Linux	No
Apache	2.4.41	Kali Linux	No
Ngnix	1.16.1	Kali Linux	No

Lastly, in the preceding approaches, we notice that network-based solutions require administrative work, expert knowledge, and their robustness is questionable, *e.g.,* the SSL method or Host header. On the other side, browser-based approaches offer a better level of security, *e.g.,* INP or eSOP.

4 FAIL-REBIND

The aim of this section is to introduce our proposed approach for halting DNS rebinding attacks and compare our approach with preceding approaches. Given the lack of robust protection in widely-used browsers, we have developed a plug-in, named Fail-rebind, for the Chrome and Firefox browsers [7]. DNS rebinding is launched on browsers, even though it goes through all network devices, *e.g.,* firewalls, routers. Layers of encryption render packet inspection infeasible at the network. On the other hand, analyzing requests on browsers to halt DNS rebinding is fast and not complicated, which can benefit the user regardless of the customized network devices. In the following, we first explain Fail-rebind's approach in order to detect and halt DNS rebinding attacks, and then evaluate its effectiveness in two different scenarios as well as comparing Fail-rebind with preceding approaches, discussed in section 3.

4.1 Approach

We used the web request API in Chrome and Firefox in order to observe and analyze the traffic of each tab in real-time. The API works similarly and has a common life cycle for a successful request in both browsers. Fail-rebind employs the following three events:

[7]https://github.com/Microsvuln/DNS-Rebinding-Attack-Prevention

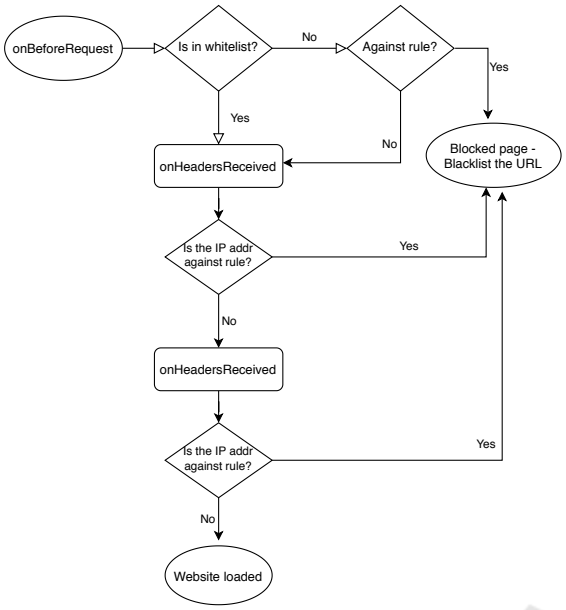

Figure 2: The Fail-rebind's workflow.

— *onBeforeRequest*: This event is fired when a request is about to be made, which can be blocked or redirected.

— *onHeadersReceived*: This event is fired when an HTTP(S) response header is received and enables extensions to add, modify, and delete response headers. Canceling a request is possible in this event.

— *onResponseStarted*: This event is triggered when the first byte of the response body is received. In this event, modifying and canceling the request is not possible.

Figure 2 depicts the workflow of the Fail-rebind plug-in. Fail-rebind consists of two components, namely IP scanning, and DNS rebinding detection. The IP scanning component first detects every attempt at private IP scanning. The user is able to whitelist his trusted local domains; therefore, the IP scanning detection module ensures that requests are initiated from untrusted domains and do not point to any private IP ranges. If such requests are to be made, they will be blocked in the *onBeforeRequest* event and the corresponding website will be blacklisted.

To stop DNS rebinding attacks, we must obtain the IP address of a request alongside its domain name. This is accomplished slightly differently in the two browsers due to the behavior of the web request API. In Chrome, the IP address of a request can be obtained in the *onResponseStarted* event, while in Firefox it is in the *onHeadersReceived* event. If the obtained IP address complies with private IP range rules, the corresponding tab will be redirected to a custom block page and the website will be blacklisted.

Table 4: Comparison between fail-rebind and other approaches.

	Desktop mobile	No Expert Knowledge	Network independent	Location independent
Fail-rebind	✓	✓	✓	✓
SSL	✓	×	×	×
HTTP Header	✓	×	×	×
Security Proxy	✓	×	×	×
dnswall	✓	×	×	×
Brahmasani et al	×	×	×	×
eSOP	×	✓	✓	✓
INP	×	✓	✓	✓

4.2 Evaluation

To evaluate Fail-rebind, we assess the two plausible scenarios explained in subsection 2.2. We arm the Firefox browser with the Fail-rebind plug-in and navigate to the malicious website, *i.e., attacker.xyz*. Given that a VPN is used by the user, the malicious website needs to scan private IP address ranges. When *attacker.xyz* establishes the first request to *192.168.0.1*, Fail-rebind evaluates the request in the onBeforeRequest event. As the website and its request are not in the whitelist, Fail-rebind blocks the request and adds the website to its blacklist. Finally, the user is informed of the attack and access to the website is denied.

In another scenario, our assumption is that the *attacker.xyz* website is already aware of the user's private IP address range. Therefore, the IP address of the PWS system (*i.e.,* 192.168.10.5) is set as the second IP address on the attacker's DNS server. When the malicious page establishes continuous requests to *attacker.xyz* to defeat DNS pinning in Firefox, Fail-rebind checks each request's IP address in the onHeadersReceived event. The moment Firefox receives the new IP address (*i.e.,* 192.168.10.5), Fail-rebind evaluates the IP address against the built-in rules. As the received IP address is private, Fail-rebind blocks the request, blacklists the *attacker.xyz* address, and warns the user with a customized error page. Similar to other existing measures, our approach can afford protection against DNS rebinding attacks targeted only at private nodes (firewall circumvention) not at public nodes (explained in subsection 2.1). We compared Fail-rebind to other approaches from four significant perspectives, explained in Table 4. The major advantage of our proposed solution is its location-independence compared to some of the previous methods, *e.g.,* SSL, HTTP header, and dnswall. This feature allows for abundant flexibility in surfing websites in any location without jeopardizing the private network's nodes. Moreover, our protection mechanism as well as other approaches. *e.g.,*

SSL, can provide safety for users both in mobile and desktop browsers. However, in many scenarios, the victim cannot request all the nodes in their private network to install SSL on their websites but Fail-rebind protects the victim irrespective of the security of the nodes in the internal network. A major advantage of using browser-based solutions, including Fail-rebind, eSOP, and INP, is that they do require neither expert knowledge nor special network devices to be in place. Lastly, however, a user may prefer browsers where plug-in development is not yet supported, and accordingly, the user's activity on one platform may still have deleterious effects on its network's nodes. We also contacted the Firefox developers and shared our findings. We have discussed possible solutions for implementing our prototype as a preventive measure inside the browser.

5 CONCLUSION

As far as user privacy is concerned, the importance of halting the DNS rebinding attack cannot be overstated. We first showed that this attack can adversely affect internal nodes. Then, we evaluated the current prevention systems and discussed their weaknesses. We also proposed a browser-based plug-in for two popular browsers, *i.e.,* Chrome and Firefox. Our approach functions perfectly under an encrypted connection, detects DNS rebinding attacks at its initial stages, and does not suffer from the drawbacks of previous systems. However, as the proposed solution is a browser-specific solution, a similar approach needs to be developed and incorporated into browsers as an internal component.

ACKNOWLEDGMENT

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "Agile Software Assistance" (SNSF project No. 200020-181973, Feb. 1, 2019 – April 30, 2022).

REFERENCES

Afek, Y., Bremler-Barr, A., and Noy, A. (2019). Eradicating attacks on the internal network with internal network policy. *arXiv preprint arXiv:1910.00975*.

Brahmasani, S. and Sivasankar, E. (2013). Two level verification for detection of DNS rebinding attacks. *International Journal of System Assurance Engineering and Management*, 4(2):138–145.

Dai, Y. and Resig, R. (2013). FireDrill: Interactive DNS rebinding. In *Presented as part of the 7th USENIX Workshop on Offensive Technologies*.

Dean, D., Felten, E. W., and Wallach, D. S. (1996). Java security: From HotJava to Netscape and beyond. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, pages 190–200. IEEE.

Fainelli, F. (2008). The OpenWrt embedded development framework. In *Proceedings of the Free and Open Source Software Developers European Meeting*, page 106. sn.

Hazhirpasand, M. and Ghafari, M. (2018). One leak is enough to expose them all. In *International Symposium on Engineering Secure Software and Systems*, pages 61–76. Springer.

Heffner, C. (2010). Remote attacks against SOHO routers. *Blackhat USA*.

Jackson, C., Barth, A., Bortz, A., Shao, W., and Boneh, D. (2009). Protecting browsers from DNS rebinding attacks. *ACM Transactions on the Web (TWEB)*, 3(1):1–26.

Johns, M., Lekies, S., and Stock, B. (2013). Eradicating DNS rebinding with the extended same-origin policy. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, pages 621–636.

Lalia, S. and Moustafa, K. (2019). Implementation of web browser extension for mitigating CSRF attack. In *World Conference on Information Systems and Technologies*, pages 867–880. Springer.

Pandiaraja, P. and Parasuraman, S. (2015). Applying secure authentication scheme to protect DNS from rebinding attack using proxy. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*, pages 1–6. IEEE.

Rekhter, Y., Moskowitz, B., Karrenberg, D., Groot, G. d., and Lear, E. (1996). Rfc1918: Address allocation for private internets.

Tatang, D., Suurland, T., and Holz, T. (2019). Study of DNS rebinding attacks on smart home devices. In *Computer Security*, pages 391–401. Springer.

Van Goethem, T., Chen, P., Nikiforakis, N., Desmet, L., and Joosen, W. (2014). Large-scale security analysis of the web: Challenges and findings. In *International Conference on Trust and Trustworthy Computing*, pages 110–126. Springer.