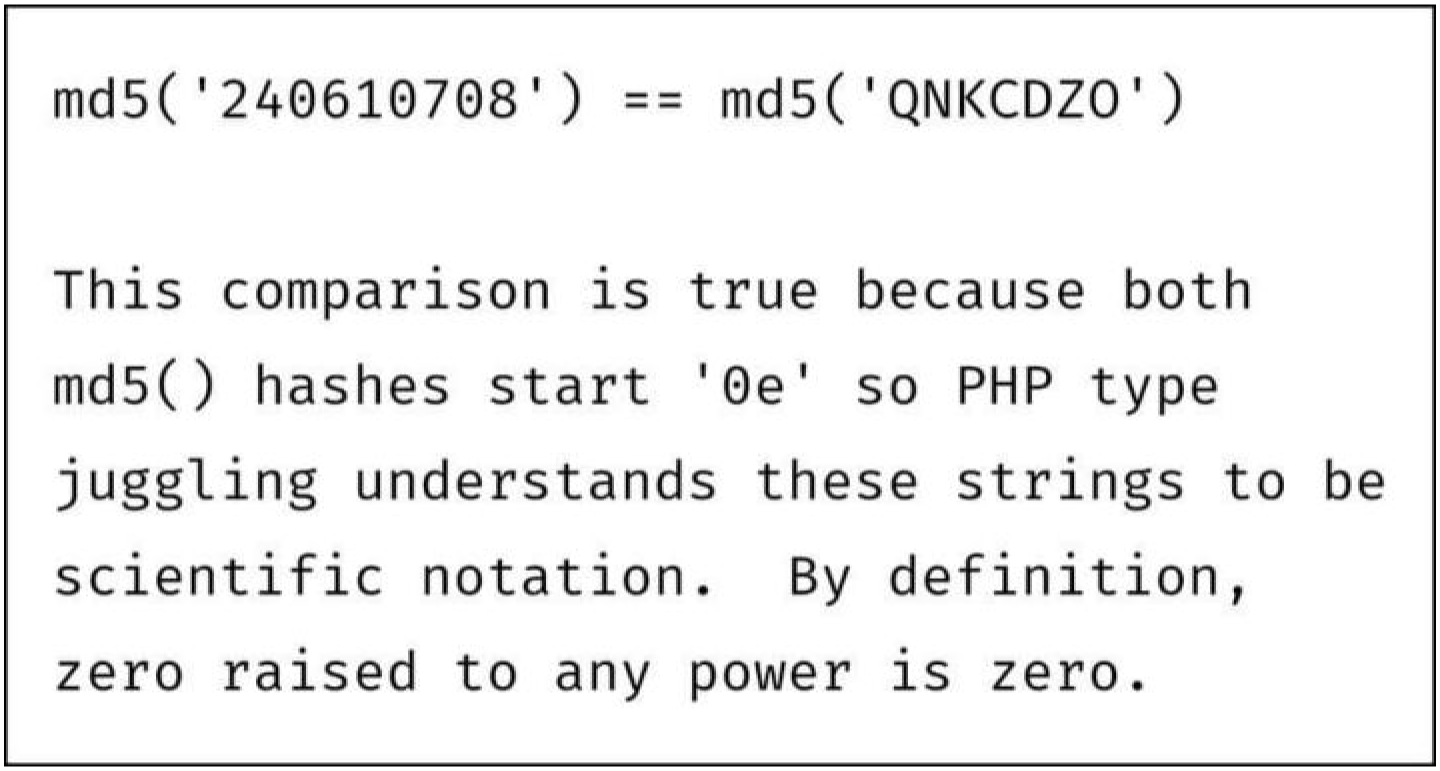
Security Now! #744 - 12-10-19
VPN-geddon Denied

This week on Security Now!

This week we look at Microsoft's force-feeding of Windows 10 feature updates, the creation of a tool to keep Win7 & 8 updates freely flowing for free, the continuing evolution of a new highly secure programming language, an update to Microsoft's RDP client for iOS, Avast & AVG in the doghouse, some VERY severe authentication bypasses in OpenBSD, a note about the WireGuard VPN... and then we take a look at the report which every security website breathlessly covered... and got wrong.

One of our stories this week is about the evolution of a new language designed with security foremost. The following picture of the week shows a beautiful example of just how anti-secure a language (PHP) can be…



```
md5('240610708') == md5('QNKCDZO')
```

This comparison is true because both md5() hashes start '0e' so PHP type juggling understands these strings to be scientific notation. By definition, zero raised to any power is zero.

Security News

Let's kick off with some Microsoft news...

Microsoft has started forcing feature updates on people who apparently don't want them. Windows 10, version 1809 first became available more than a year ago on November 13, 2018. And that release of Windows for Home, Pro, Pro Education, and Pro for Workstations editions has an end of service date of May 12, 2020. After May 12, 2020, devices still running the October 2018 Update will no longer receive any future quality and security updates.

Therefore, in a planned preemptive move to keep users of that Windows edition from falling out of maintenance, last Thursday, Microsoft announced: *"Beginning today, we will slowly start the phased process to automatically initiate a feature update for devices running the October 2018 Update Home and Pro editions. Keeping those devices supported and receiving the monthly updates that are critical to device security and ecosystem health. We are starting this rollout process several months in advance of the end of service date to provide adequate time for a smooth update process."*

So, if you or people you know have, for whatever reason, been forestalling the move from the 1809 edition of Windows 10, you should not be surprised when Windows will no longer accept "thanks anyway, but not now" as the answer about whether you want the latest and greatest Windows 10. You will soon be getting this autumn's 1909 edition whether you want it or not.

I think that given how problematical release 1809 had been, causing its roll-out to be delayed well into this year, and as we said at the time, it made absolute sense to hold off. Especially if neither the following spring nor fall feature updates were particularly tantalizing.

Bypass to continue obtaining Win7 updates created.
In a bit of "we should definitely have seen this coming" news, a group of Windows enthusiasts who hang out over in the "My Digital Life" forum have come up with a "BypassESU" utility where "ESU" stands for "Extended Security Updates."

https://forums.mydigitallife.net/threads/bypass-windows-7-extended-security-updates-eligibility.80606/

Two months before it'll be needed, we already have a hack to trick Windows 7 and 8 into continuing to receive another three years of security updates after the otherwise final January 14th final Patch Tuesday.

So the story goes... Last month, Microsoft released a test Windows 7 ESU update (KB4528069) to allow IT administrators to verify that their systems would be compatible with the upcoming ESU (Extended Security Updates) process.

https://support.microsoft.com/en-us/help/4528069/update-for-eligible-windows-7-and-server-2008-r2-devices-can-get-esu

Summary

This optional nonsecurity update will help you verify that your eligible Windows 7 Service Pack 1 (SP1) and Server 2008 R2 SP1 devices can continue to get Extended Security Updates (ESUs) after the end of support date of January 14, 2020.

Notes

- This update has no actual security content. This update is a test package we recommend that you deploy in your test environment. Install this update on your on-premise devices that are eligible for ESU.
- This update can be installed on x64-based architecture for Windows Server 2008 R2 SP1 and the Windows editions mentioned in Lifecycle FAQ-Extended Security Updates.
- This update can be installed on x86 and x64-based architecture for Windows 7 SP1 and the Windows editions mentioned in Lifecycle FAQ-Extended Security Updates.
- This update is not applicable for Windows 7 Virtual Desktop (WVD) and Windows 7 Embedded OS.
- Installing this update has no impact on getting security updates between now and January 14, 2020.
- For technical support, customers must purchase Extended Security Updates and have an active support plan in place to get technical support on a product that has moved beyond the Extended Support date. Please call 1-800-Microsoft (642-7676) to receive support.

Installation instructions

- Install this update from one of the Release Channels such as Microsoft Update Catalog or Windows Server Update Services.
- Verify the successful installation of this update by using your regular process such as Update History.

Prerequisites

You must have the following installed on your on-premise device before you apply this update:

- Install the following SHA-2 code signing support update and servicing stack update (SSU) or a later SSU update:
- 4474419 SHA-2 code signing support update for Windows Server 2008 R2, Windows 7, and Windows Server 2008: September 23, 2019
- 4490628 Servicing stack update for Windows 7 SP1 and Windows Server 2008 R2 SP1: March 12, 2019
- Install the following servicing stack update (SSU) and monthly rollup:
- 4516655 Servicing stack update for Windows 7 SP1 and Server 2008 R2 SP1: September 10, 2019
- 4519976 October 8, 2019—KB4519976 (Monthly Rollup)
- Install and activate the ESU key. For information about how to install and activate the ESU key, see the How to get Extended Security Updates for eligible Windows devices blog on the Microsoft Tech Community website.

Note After activation, you can then continue to use your current update and servicing strategy to deploy ESU through Windows Update, Windows Server Update Services (WSUS), or whichever update management solution that you prefer.

That last prerequisite about “installing and activating the ESU key” is the trick.

In an echo of the ways Windows modders managed to bypass Windows OS installation key checks for decades, the community at My Digital Life created a “BypassESU” tool to circumvent the ESU key check operation and enable the installation of the test ESU… and all of the future updates that it allows.

However, we can also expect this victory to be short lived. It’ll likely be more reminiscent of all past iOS jailbreaks, which were immediately fixed the instant they were discovered, than the permanent CheckM8 bootrom break which can never be fixed. In other words, Microsoft seems sure to counter that hack with an update -- which, in an ironic twist, is exactly what the hack is designed to permit. Such an update would return any ESU Bypassed machines to their original “no more updates for you!” state. It’ll be fun to watch this and see how this back-and-forth develops.

Meanwhile, Microsoft's Project Verona continues moving forward.
Microsofties are talking move and more about Verona, which is the project codename for Microsoft's increasingly customized, tightened and hardened language which is reminiscent of and has been strongly influenced by the RUST programming language.

So, to first get an update on where Microsoft is with RUST, Adam Burch, a software engineer in the Hyper-V team wrote the following last month to the MSRC -- Microsoft Security Response Center -- blog: https://msrc-blog.microsoft.com/2019/11/07/using-rust-in-windows/

Using Rust in Windows
Tags: Security Research & Defense / By MSRC Team / November 7, 2019 / Memory Safety, Rust, Safe Systems Programming Languages, Secure Development

This Saturday 9th of November, there will be a keynote from Microsoft engineers Ryan Levick and Sebastian Fernandez at RustFest Barcelona. They will be talking about why Microsoft is exploring Rust adoption, some of the challenges we’ve faced in this process, and the future of Rust adoption in Microsoft. If you want to talk with some of the people working on how Microsoft is evolving its code practices for better security, be sure to attend the keynote and talk to Ryan and Sebastian afterwards!

This blog describes part of the story of Rust adoption at Microsoft. Recently, I’ve been tasked with an experimental rewrite of a low-level system component of the Windows codebase (sorry, we can’t say which one yet). Instead of rewriting the code in C++, I was asked to use Rust, a memory-safe alternative. Though the project is not yet finished, I can say that my experience with Rust has been generally positive. It’s a good choice for those looking to avoid common mistakes that often lead to security vulnerabilities in C++ code bases.
Great dev experience

For C++ developers used to writing complex systems, using Rust as a developer is a breath of fresh air. The memory and data safety guarantees made by the compiler give the developer much greater confidence that compiling code will be correct beyond memory safety vulnerabilities. Less time is spent debugging trivial issues or frustrating race conditions. The compiler warning and error messages are extremely well written, allowing novice Rust programmers to quickly identify and resolve issues in their code. Visual Studio Code already has a helpful extension (RLS), which provides Intellisense suggestions and syntax highlighting. Additionally, the Cargo build tool offers very helpful features around testing, documentation generation, and auto formatting.

Learning Curve
Thanks to a plethora of online documentation and very helpful compiler error messages, Rust has a pretty easy learning curve for someone like me who has used C++ for the majority of my career. There are tutorials aimed specifically at C/C++ systems engineers. In his talk at RustConf 2019, Jeremy Fitzhardinge at Facebook noted that he saw experienced C/C++ developers become comfortable with Rust in around four weeks and pretty fluent in eight. This aligns with my own experience. I participated in the annual Microsoft internal "One Week" hackathon with one experienced Rust developer and one complete novice. Within three days, the novice Rust developer had written more than 1000 lines of idiomatic Rust code. In addition to the great documentation, there are helpful tools like Clippy which allow experienced C++ developers to jump right into coding Rust without much direct assistance from those experienced with Rust.

As we expand the use of Rust inside of Microsoft, I believe it will be prudent to start a RustReviewers group for any PRs that include Rust code. This will help novices in diverse teams get feedback from Rust experts, regardless of the specific problem domain.

In general, new components or existing components with clean interfaces will be the easiest to port to Rust. The component I've been rewriting has been challenging as there are many abstractions leaked from one layer to the next, requiring some preliminary refactoring before progress could be made.

Keeping it Safe
To obtain the desired safety guarantees from Rust, strict guidelines must be placed around the use of the unsafe keyword. Any calls to a Foreign Function Interface (FFI) function should occur in a wrapper function that provides a safe abstraction around it. Similarly, any other code that must use the unsafe keyword should have a wrapper function or struct that provides a safe abstraction.

In practice, besides FFI boundaries, unsafe has only been required for very basic protocol handling. In these cases, it's simple to write some generic unsafe code that can be thoroughly unit tested and used for a variety of scenarios, resulting in code that feels much safer than C++. After writing Rust code, I find writing C++ much more frustrating since I can't rely on the compiler to ensure memory safety.

Beyond just ensuring safety guarantees, having a set of internal Rust coding standards will help new developers get the most out of the language. Best practices for error handling, logging, locking, and other language-specific issues will obtain higher quality code faster.

Desired features and the Rust community
Given Rust's relative youth, there are still some missing language features that would be very desirable for our development. Foremost of these are safe transmutation (safely cast "plain old data" types to and from raw bytes), safe support for C style unions, and fallible allocation (fail gracefully from allocation failure, rather than panic).

Another amazing capability from Rust is the unit testing built into Cargo which allows developers to write unit tests in the same file as production code and run them easily while developing. Unfortunately, we cannot rely on Cargo as a build tool today inside of our large and complex build system, so we cannot rely on these tests in automatic code integration flows without further work. Talk has already started with the Cargo team on how large companies like Microsoft with complex existing build systems can still take advantage of Cargo.

Due to the interest in using Rust in microcontrollers and low-level systems like kernels and hypervisors, the community is already working in solving these issues. I'm confident that we at Microsoft will be able to help in these endeavors to shape the future of the language to improve its usefulness in these scenarios.

Going from Here
In general, using Rust has been a really great experience. I look forward to seeing more developers at Microsoft working with the language and working with the wider community on making the language an even better fit for some of the things we do here at Microsoft.

In other words, Microsoft is clearly investing some significant resources into RUST and will be sharing forward and putting everything they develop back into the open source RUST community. One of the things that Adam's posting reveals is that it's very different for a single developer to use a new language such as RUST versus an organization like Microsoft with a mammoth code base that requires a meta-level of automated code management oversight. But based up Adam's posting, this appears to be a direction Microsoft is seriously considering.

As I've been saying recently, we now have SUCH an excess of computational power, RAM and non-volatile storage, and we have so much experience with software flaws staring us in the face, that we really can finally afford to take a serious step back and deeply rethink our programming language choices to enlist the help of the languages we're using to prevent tomorrow's security flaws. We really do need to fix the way things have been done. It's clearly time. And since everything is economics, I'm sure that Microsoft can see the true economic benefits arising from having a far more secure product offering in the future.

Microsoft's RDP client for iOS is back
I don't think I was aware that Microsoft had a Remote Desktop Protocol (RDP) client for iOS. That might be sort of cool, though.

Their RDP client for iOS had not been updated for more than a year. Then a couple of weeks ago they updated the client to v10, put it up on the Apple App Store... then almost immediately pulled it down without saying anything.

Based upon some hearsay from some users, the short-lived v10.0 Remote Desktop Client would wipe all of the user's saved Remote Desktop settings. One user posted: "Its because it wipes all your saved config inside the app - it resets it to new - I managed to update before they pulled it" Another user also encountered a similar issue, but after force-closing the app and restarting it, their profiles were available.

Then, last Wednesday, the iOS RDP client reappeared as v10.0.1:
https://apps.apple.com/fi/app/microsoft-remote-desktop/id714464092

Microsoft Remote Desktop / Get work done from anywhere / Microsoft Corporation
Use Microsoft Remote Desktop for iOS to connect to a remote PC or virtual apps and desktops made available by your admin. With Microsoft Remote Desktop, you can be productive no matter where you are.
KEY FEATURES

- Access remote PCs running Windows Professional or Enterprise and Windows Server.
- Access managed resources published by your admin. Connect through a Remote Desktop Gateway.
- Rich multi-touch experience supporting Windows gestures.
- Secure connection to your data and applications.
- Simple management of your connections and user account from the Connection Centre.
- Audio and video streaming.
- Redirect your clipboard and local devices such as microphones and cameras.

What's New / 3 Dec 2019

Version 10.0.1

It's been well over a year since we last updated the Remote Desktop Client for iOS. However, we are back in the game with an exciting update, with many more to follow at a regular cadence from here on out. Here's what new in this release:

- Support for the Windows Virtual Desktop (WVD) service.
- Brand new Connection Centre UI.
- Brand new in-session UI for switching between connected PCs and apps.
- New layout for the auxiliary on-screen keyboard.
- Improved external keyboard support.
- Support for SwiftPoint Bluetooth mice.
- Support for microphone redirection.
- Support for local storage redirection.
- Support for camera redirection (Windows 10 1809 or later required).
- Support for new iPhone and iPad devices.
- Dark and light theme support.
- Control whether your phone can lock when connected to a remote PC or app.
- Collapse the in-session connection bar with a long-press on the Remote Desktop logo.

We're committed to making this app the best Remote Desktop client for iOS and value your feedback. If you encounter any errors, you can contact us via Settings > Report an Issue.

I'm going to have to make some time to give that a try and play with it. The combination of an iOS VPN for RDP access control and security, and this RDP client for iOS is quite compelling. And since it can be used with a Bluetooth mouse with the iPad… that sounds pretty cool!

(It turns out that iPadOS quietly added mouse support and didn't make a big deal about it. It needs to be enabled and is under the "Assistive Support" menu of the iPad Control applet.)

Okay… enough Microsoft stuff…

Avast / AVG in the doghouse
The Avast Online Security, Avast SafePrice, AVG Online Security, and AVG SafePrice extensions have all been pulled from the Mozilla addon repository... after they were found to be silently collecting FAR more user-identifiable data than was required to do their jobs.

The story begins with a German web researcher and developer by the name of Wladimir Palant. Does Wladimir have any street cred? I'd say so. The first two paragraphs of his Bio read:

> My name is Wladimir Palant and developing software is both my day job and a hobby. I became fascinated by Firefox extension development in 2003, and my Adblock Plus extension became so popular that eventually I co-founded Eyeo GmbH to continue its development. By now Adblock Plus isn’t available just for Firefox but for Chrome, Opera, Safari, Android and Internet Explorer as well. My other notable Firefox extensions are Google/Yandex search link fix as well as developer tools JavaScript Deobfuscator and Extension Auto-Installer.
>
> My other area of interest is web application security. There is a number of ways in which web application can be vulnerable to attacks, and most web application developers aren’t terribly familiar with those — or they don’t know a good way to mitigate the attacks. I’ve been looking into locating vulnerabilities like XSS, CSRF, SQL Injection and others in large websites and reported those to the website owners. I’ve also published a number of articles with recommendations on writing secure code.

So, Wladimir was poking around the Avasts & AVG extensions and was shocked by what he found. This announcement and take down is titled: “Avast Online Security and Avast Secure Browser are spying on you.”

https://palant.de/2019/10/28/avast-online-security-and-avast-secure-browser-are-spying-on-you/

Are you one of the allegedly 400 million users of Avast antivirus products? Then I have bad news for you: you are likely being spied upon. The culprit is the Avast Online Security extension that these products urge you to install in your browser for maximum protection.

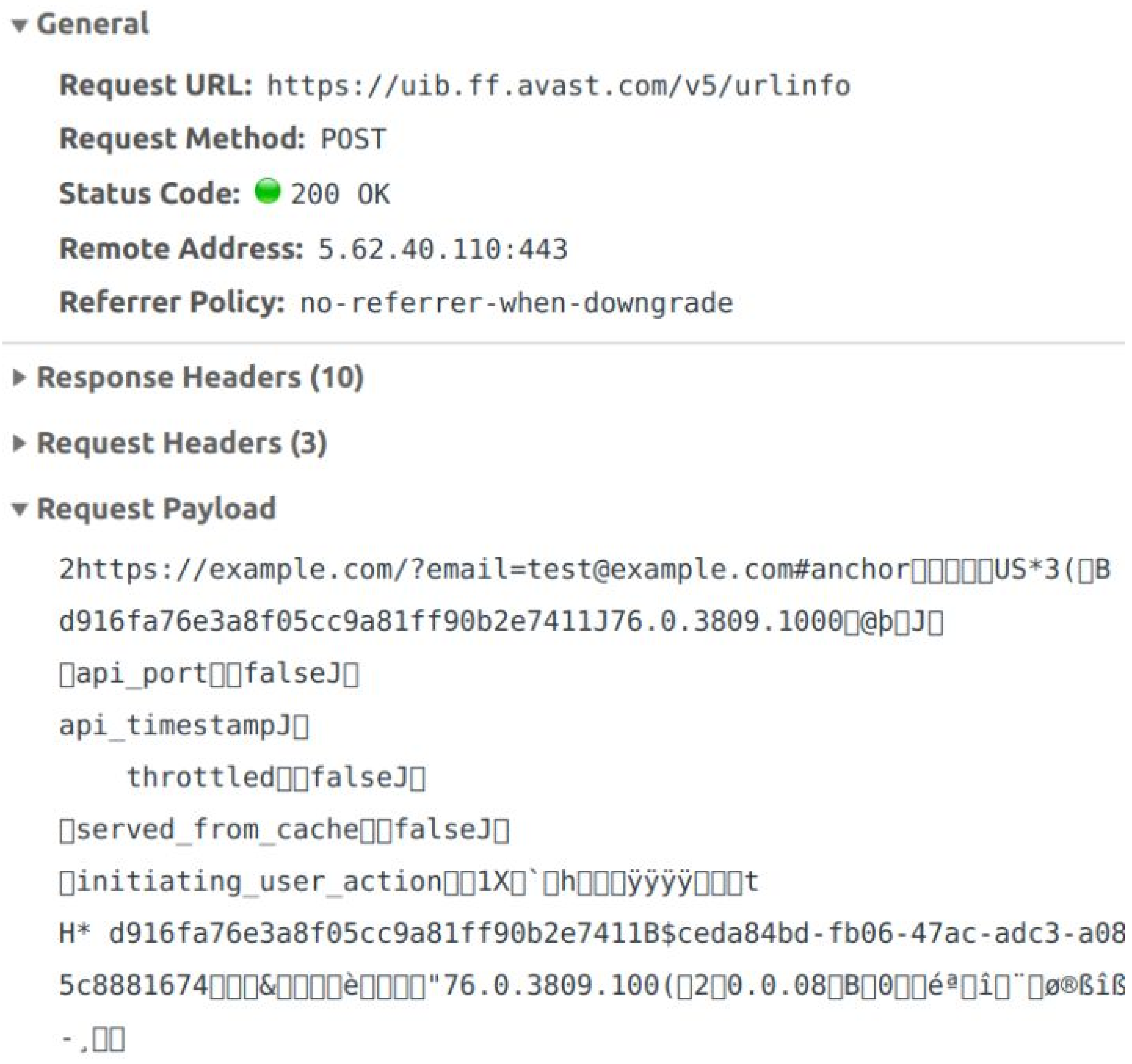
But even if you didn’t install Avast Online Security yourself, it doesn’t mean that you aren’t affected. This isn’t obvious, but Avast Secure Browser has Avast Online Security installed by default. It is hidden from the extension listing and cannot be uninstalled by regular means, its functionality is apparently considered an integral part of the browser. Avast products promote this browser heavily, and it will be used automatically in “Banking Mode.” Given that Avast bought AVG a few years ago, there is also a mostly identical AVG Secure Browser with the built-in AVG Online Security extension.

Summary of the findings
When Avast Online Security extension is active, it will request information about your visited websites from an Avast server. In the process, it will transmit data that allows reconstructing

your entire web browsing history and much of your browsing behavior. The amount of data being sent goes far beyond what's necessary for the extension to function, especially if you compare to competing solutions such as Google Safe Browsing.

Avast Privacy Policy covers this functionality and claims that it is necessary to provide the service. Storing the data is considered unproblematic due to anonymization (I disagree), and Avast doesn't make any statements explaining just how long it holds on to it.

What is happening exactly?
Using browser's developer tools you can look at an extension's network traffic. If you do it with Avast Online Security, you will see a request to https://uib.ff.avast.com/v5/urlinfo whenever a new page loads in a tab:



▾ General

```
Request URL: https://uib.ff.avast.com/v5/urlinfo
Request Method: POST
Status Code: ● 200 OK
Remote Address: 5.62.40.110:443
Referrer Policy: no-referrer-when-downgrade
```

▸ Response Headers (10)

▸ Request Headers (3)

▾ Request Payload

```
2https://example.com/?email=test@example.com#anchor□□□□□US*3(□B
d916fa76e3a8f05cc9a81ff90b2e7411J76.0.3809.1000□@þ□J□
□api_port□□falseJ□
api_timestampJ□
    throttled□□falseJ□
□served_from_cache□□falseJ□
□initiating_user_action□□1X□`□h□□□ÿÿÿ□□□t
H* d916fa76e3a8f05cc9a81ff90b2e7411B$ceda84bd-fb06-47ac-adc3-a08
5c8881674□□□&□□□□è□□□□"76.0.3809.100(□2□0.0.08□B□0□□éª□î□¨□ø®ßîß
-¸□□
```

So the extension sends some binary data and in return gets information on whether the page is malicious or not. The response is then translated into the extension icon to be displayed for the page. You can clearly see the full address of the page in the binary data, including query part and anchor. The rest of the data is somewhat harder to interpret, I'll get to it soon.

This request isn't merely sent when you navigate to a page, it also happens whenever you switch tabs. And there is an additional request if you are on a search page. This one will send every single link found on this page, be it a search result or an internal link of the search engine.

What data is being sent?
The binary UrlInfoRequest data structure used here can be seen in the extension source code. It is rather extensive however, with a number of fields being nested types. Also, some fields appear to be unused, and the purpose of others isn't obvious. Finally, there are "custom values" there as well which are a completely arbitrary key/value collection. That's why I decided to stop the extension in the debugger and have a look at the data before it is turned into binary. If you want to do it yourself, you need to find this.message() call in scripts/background.js and look at this.request after this method is called. The interesting fields were:

Field	Contents
uri	The full address of the page you are on.
title	Page title if available.
referer	Address of the page that you got here from, if any.
windowNum tabNum	Identifier of the window and tab that the page loaded into.
initiating_user_action windowEvent	How exactly you got to the page, e.g. by entering the address directly, using a bookmark or clicking a link.
visited	Whether you visited this page before.
locale	Your country code, which seems to be guessed from the browser locale. This will be "US" for US English.
userid	A unique user identifier generated by the extension (the one visible twice in the screenshot above, starting with "d916"). For some reason this one wasn't set for me when Avast Antivirus was installed.
plugin_guid	Seems to be another unique user identifier, the one starting with "ceda" in the screenshot above. Also not set for me when Avast Antivirus was installed.
browserType browserVersion	Type (e.g. Chrome or Firefox) and version number of your browser.
os osVersion	Your operating system and exact version number (the latter only known to the extension if Avast Antivirus is installed).

And that's merely the fields which were set. The data structure also contains fields for your IP address and a hardware identifier but in my tests these stayed unused. It also seems that for paying Avast customers the identifier of the Avast account would be transmitted as well.

What does this data tell about you?
The data collected here goes far beyond merely exposing the sites that you visit and your search history. Tracking tab and window identifiers as well as your actions allows Avast to create a nearly precise reconstruction of your browsing behavior: how many tabs do you have open, what websites do you visit and when, how much time do you spend reading/watching the contents, what do you click there and when do you switch to another tab. All that is connected to a number of attributes allowing Avast to recognize you reliably, even a unique user identifier.

If you now think "but they still don't know who I am" – think again. Even assuming that none of the website addresses you visited expose your identity directly, you likely have a social media account. There have been a number of publications showing that, given a browsing history, the corresponding social media account can be identified in most cases.

Isn't this necessary for the extension to do its job?
No, the data collection is definitely unnecessary to this extent. You can see this by looking at how Google Safe Browsing works, the current approach being largely unchanged compared to how it was integrated in Firefox 2.0 back in 2006. Rather than asking a web server for each and every website, Safe Browsing downloads lists regularly so that malicious websites can be recognized locally:

> No information about you or the sites you visit is communicated during list updates. […] Before blocking the site, Firefox will request a double-check to ensure that the reported site has not been removed from the list since your last update. This request does not include the address of the visited site, it only contains partial information derived from the address.

[[[So Firefox/Google Safe Browsing is deliberately protecting their users' privacy while Avast/AVG are doing quite the opposite.]]]

"I've seen a bunch of similar extensions by antivirus vendors, and so far all of them provided this functionality by asking the antivirus app. Presumably, the antivirus has all the required data locally and doesn't need to consult the web service every time."

Edit (2019-10-29): I got a hint that Avast acquired Jumpshot a bunch of years ago. And if you take a look at the Jumpshot website, they list "Incredibly detailed clickstream data from 100 million global online shoppers and 20 million global app users. Analyze it however you want: track what users searched for, how they interacted with a particular brand or product, and what they bought. Look into any category, country, or domain." So you now have a pretty good guess as to where your data is going.

Conclusions
Avast Online Security collecting personal data of their users is not an oversight and not necessary for the extension functionality either. The extension attempts to collect as much context data as possible, and it does so on purpose. The Avast privacy policy shows that Avast is

aware of the privacy implications here. However, they do not provide any clear retention policy for this data. They rather appear to hold on to the data forever, feeling that they can do anything with it as long as the data is anonymized. The fact that browsing data can usually be deanonymized doesn't instill much confidence however.

This is rather ironic given that all modern browsers have phishing and malware protection built in that does essentially the same thing but with a much smaller privacy impact. In principle, Avast Secure Browser has this feature as well, it being Chromium-based. However, all Google services have been disabled and removed from the settings page – the browser won't let you send any data to Google, sending way more data to Avast instead.

Wladimir reported what he found to Mozilla, who agreed, and immediately pulled all of those Avast and AVG extensions from the Firefox extension repository. Avast have stated that they are working to correct this and will have an acceptable extension -- which collects far less information -- available as soon as possible.

MY annoyance with this behavior is that companies are essentially doing everything they can get away with. This is not the end of the world for any of Avast's 400 million users. But they appear to have been collecting all of this data just because they could. No one said no. No one had looked … and it's just not possible for every piece of communicating software to be examined and vetted.

Also note that Google was also informed but has so far not acted. All four extensions are reportedly still available from the Google Chrome Web Store but Wladimir believes that they will be removed after "considerable news coverage." It sure does seem as though they should be.

The WireGuard VPN
In our final piece I'll be mentioning, in passing, a very compelling new VPN solution which I haven't yet made time to discuss in depth, and that's "WireGuard." I won't be referring to it because there's anything wrong with it -- quite the opposite, really. I am not alone in believing that the future of VPNs is something called "WireGuard."

I first became aware of it last summer and after reading into it a bit I bookmarked the project for follow-up to learn everything about it, so I could tell everyone on this podcast. It was EVERYTHING I loved: A complete reconceptualization and rewrite, from scratch, of a VPN (very much as I was planning to do with CryptoLink). A SIMPLE take on the VPN. In their own words:

"WireGuard is an extremely simple yet fast and modern VPN that utilizes state-of-the-art cryptography. It aims to be faster, simpler, leaner, and more useful than IPsec, while avoiding the massive headache. It intends to be considerably more performant than OpenVPN. WireGuard is designed as a general purpose VPN for running on embedded interfaces and super computers alike, fit for many different circumstances. Initially released for the Linux kernel, it is now cross-platform (Windows, macOS, BSD, iOS, Android) and widely deployable. It is currently under heavy development, but already it might be regarded as the most secure, easiest to use, and simplest VPN solution in the industry."

Some time after that, while doing research for the podcast, I encountered an independent set of benchmarks comparing WireGuard with OpenVPN and it wasn't even close. WireGuard blew the packets off OpenVPN. I haven't switched over to using it myself only because I've had a bit too much on my plate as is. But this goes a LONG way toward solving one of the longstanding annoyances with a VPN... Which is that, as we know, it tends to be slower than a non-VPN connection. WireGuard shifts the tradeoff between security and performance.

The next time I encountered WireGuard was a total coincidence. The sponsor of my OWASP SQRL presentation in Sweden was a company named Mullvad which I had never heard of at the time. (You may notice their marketing materials standing next to me in the auditorium where I presented SQRL in the video from Sweden.) What was significant was that they, Mullvad, took Lorrie and me, Rasmus Vind (who did the SQRL XenForo integration, Daniel Persson, who started both the SQRL Android client and the SQRL Wordpress plugin) and the Swedish OWASP guys, all out to dinner the night before. And I learned that Mullvad is a VERY COOL company. They are exclusively a WireGuard VPN endpoint service, so I was really interested to learn about their experiences with WireGuard. I came away VERY impressed because they bend over backwards to never know who their own customers are. To sign up, you go to their site and obtain a long random serial number. That's your identified. No name, no eMail, nothing else. You just get a number from them. Then, one way or another, you arrange to provide anonymous funding to them crediting that serial number. They told hilarious stories about receiving boxes of coins from completely unknown sources. Accepting metallic cash is a royal pain in the butt, but they do it. They count the coins and credit the indicated account.

I met the company's owner. He picked up the tab for dinner, and I was very impressed by how truly serious he was about enforcing the privacy of their users... None of whom he knows. The fundamental right to privacy is like a religious crusade for him.

So anyway, that's the background and we will definitely be taking a deep dive into the WireGuard protocol early next year. But the reason I'm talking about it today, and the reason I needed to provide that context, is that there's the news about WireGuard and Linux (and I was surprised to hear Mullvad mentioned at the end of the article)...

Steven Vaughan-Nichols, writing for ZDNet, asked rhetorically, "how much are people looking forward to WireGuard, the new in-kernel Linux virtual private network (VPN)? Linus Torvalds recently said, "Can I just once again state my love for it and hope it gets merged soon? Maybe the code isn't perfect, but I've skimmed it, and compared to the horrors that are OpenVPN and IPSec, it's a work of art."

That's not really damning with faint praise because for Linus, this is high praise. WireGuard has now been committed to the mainline Linux kernel. While there are still tests to be made and hoops to be jumped through, it should be released in the next major Linux kernel release, 5.6, in the first or second quarter of 2020.

WireGuard has been in development for some time. It is a layer 3 secure VPN. Unlike its older rivals, which it's meant to replace, its code is much cleaner and simple. The result is a fast, easy-to-deploy VPN. While it started as a Linux project, WireGuard code is now cross-platform, and its code is now available on Windows, macOS, BSD, iOS, and Android.

It took longer to arrive than many wished because WireGuard's principal designer, Jason Donenfeld, disliked Linux's built-in cryptographic subsystem on the grounds its application programming interface (API) was too complex and difficult. He suggested it be supplemented with a new cryptographic subsystem: His own Zinc library. Many developers didn't like this. They saw this as wasting time reinventing the cryptographic well.

But Donenfeld had an important ally. Linus wrote, "I'm 1000% with Jason on this. The crypto/ model is hard to use, inefficient, and completely pointless when you know what your cipher or hash algorithm is, and your CPU just does it well directly."

In the end, Donenfeld compromised. "WireGuard will get ported to the existing crypto API. So it's probably better that we just fully embrace it, and afterward work evolutionarily to get Zinc into Linux piecemeal." That's exactly what happened. Some Zinc elements have been imported into the legacy crypto code in the forthcoming Linux 5.5 kernel. This laid the foundation for WireGuard to finally ship in Linux early next year.

WireGuard works by securely encapsulating IP packets over UDP. It's authentication and interface design has more to do with Secure Shell (SSH) than other VPNs. You simply configure the WireGuard interface with your private key and your peers' public keys, and you're ready to securely talk.

When it arrives, I expect WireGuard to quickly become the new standard for Linux VPNs. With its tiny code-size, high-speed cryptographic primitives, and in-kernel design, it should be faster than all other existing VPN technologies. WireGuard's not just fast, it's secure as well, with its support of state-of-the-art cryptography technologies such as the Noise protocol framework, Curve25519, ChaCha20, Poly1305, BLAKE2, SipHash24, and HKD.

All this is why some companies -- like Mullvad VPN -- adopted WireGuard long before it was incorporated into Linux. As Mullvad co-founder Fredrik Strömberg wrote two-years ago, "We find WireGuard beneficial for a number of reasons. Its simplistic design in few lines of code makes it easier for sysadmins and developers to integrate it correctly -- and harder for them to get it wrong." Thus, "WireGuard will move the world one step closer to our own vision -- of making mass surveillance ineffective."

So, say hi to the future of the VPN. Its name... is WireGuard.

VPN-geddon??

So, once again we have breathless headlines:

> *"New Linux Vulnerability Lets Attackers Hijack VPN Connections"*
> *"Linux Bug Opens Most VPNs to Hijacking"*
> *"New vulnerability lets attackers sniff or hijack VPN connections"*
> *"New Linux Bug Lets Attackers Hijack Encrypted VPN Connections"*
> *"Networking attack gives hijackers VPN access"*

All of those headlines sound really bad. Not one of the sites I routinely monitor for important security news missed the opportunity to say something about it. They all covered it, yet every single one of them got it wrong.

Yes, Virginia, there is a problem, but it's not anything like it's being made out to be.

Not to pick on anyone, here’s an example of the beginning of a typical story in the tech press:

A team of cybersecurity researchers has disclosed a new severe vulnerability affecting most Linux and Unix-like operating systems, FreeBSD, OpenBSD, macOS, iOS, and Android, that could allow remote 'network adjacent attackers' to spy on and tamper with encrypted VPN connections.

The vulnerability, tracked as CVE-2019-14899, resides in the networking stack of various operating systems and can be exploited against both IPv4 and IPv6 TCP streams.

Since the vulnerability does not rely on the VPN technology used, the attack works against widely implemented virtual private network protocols like OpenVPN, WireGuard, IKEv2/IPSec, and more, the researchers confirmed.

This vulnerability can be exploited by a network attacker — controlling an access point or connected to the victim's network — just by sending unsolicited network packets to a targeted device and observing replies, even if they are encrypted.

As explained by the researchers, though there are variations for each of the impacted operating systems, the vulnerability allows attackers to:

- determine the virtual IP address of a victim assigned by the VPN server,
- determine if there is an active connection to a given website,
- determine the exact seq and ack numbers by counting encrypted packets and/or examining their size, and
- inject data into the TCP stream and hijack connections.

Sure sounds like VPN-geddon to me!

https://seclists.org/oss-sec/2019/q4/122

We have discovered a vulnerability in Linux, FreeBSD, OpenBSD, MacOS, iOS, and Android which allows a malicious access point, or an adjacent user, to determine if a connected user is using a VPN, make positive inferences about the websites they are visiting, and determine the correct sequence and acknowledgement numbers in use, allowing the bad actor to inject data into the TCP stream. This provides everything that is needed for an attacker to hijack active connections inside the VPN tunnel.

This vulnerability works against OpenVPN, WireGuard, and IKEv2/IPSec, but has not been thoroughly tested against tor, but we believe it is not vulnerable since it operates in a SOCKS layer and includes authentication and encryption that happens in userspace. It should be noted, however, that the VPN technology used does not seem to matter and we are able to make all of our inferences even though the responses from the victim are encrypted, using the size of the packets and number of packets sent (in the case of challenge ACKs, for example) to determine what kind of packets are being sent through the encrypted VPN tunnel.

There are 3 steps to this attack:

1. Determining the VPN client's virtual IP address
2. Using the virtual IP address to make inferences about active connections
3. Using the encrypted replies to unsolicited packets to determine the sequence and acknowledgment numbers of the active connection to hijack the TCP session

There are 4 components to the reproduction:

1. The Victim Device (connected to AP, 192.168.12.x, 10.8.0.8)
2. AP (controlled by attacker, 192.168.12.1)
3. VPN Server (not controlled by attacker, 10.8.0.1)
4. A Web Server (not controlled by the attacker, public IP in a real-world scenario)

The victim device connects to the access point, which for most of our testing was a laptop running create_ap. The victim device then establishes a connection with their VPN provider.

The access point can then determine the virtual IP of the victim by sending SYN-ACK packets to the victim device across the entire virtual IP space (the default for OpenVPN is 10.8.0.0/24). When a SYN-ACK is sent to the correct virtual IP on the victim device, the device responds with a RST; when the SYN-ACK is sent to the incorrect virtual IP, nothing is received by the attacker.

Similarly, to test if there is an active connection for any given website, such as 64.106.46.56, for example, we send SYN or SYN-ACKs from 64.106.46.56 on port 80 (or 443) to the virtual IP of the victim across the entire ephemeral port space of the victim. The correct four-tuple will elicit no more than 2 challenge ACKs per second from the victim, whereas the victim will respond to the incorrect four-tuple with a RST for each packet sent to it.

NOTE: By "four-tuple" they mean the combination of the source IP, source port, destination IP and destination port. -- They are proposing that a connection to a remote IP can be determined by correctly guessing, or scanning to find a match for all four. While this would technically be an information disclosure, it's very expensive and very weak, and it's more like an expensive information confirmation.

Finally, once the attacker determined that the user has an active TCP connection to an external server, we will attempt to infer the exact next sequence number and in-window acknowledgment number needed to inject forged packets into the connection. To find the appropriate sequence and ACK numbers, we will trigger responses from the client in the encrypted connection found in part 2. The attacker will continually spoof reset packets into the inferred connection until it sniffs challenge ACKs. The attacker can reliably determine if the packets flowing from the client to the VPN server are challenge ACKs by looking at the size and timing of the encrypted responses in relation to the attacker's spoofed packets. The victim's device will trigger a TCP challenge ACK on each reset it receives that has an in-window sequence number for an existing connection. For example, if the client is using OpenVPN to exchange encrypted packets with the VPN server, then the client will always respond with an SSL packet of length 79 when a challenge ACK is triggered.

The attacker must spoof resets to different blocks across the entire sequence number space until one triggers an encrypted challenge ACK. The size of the spoof block plays a significant role in how long the sequence inference takes, but should be conservative as to not skip over the receive window of the client. In practice, when the attacker thinks it sniffs an encrypted challenge-ACK, it can verify this is true by spoofing X packets with the same sequence number. If there were X encrypted responses with size 79 triggered, then the attacker knows for certain it is triggering challenge ACKs (at most 2 packets of size 79 per second).

After the attacker has inferred the in-window sequence number for the client's connection, they can quickly determine the exact sequence number and in-window ACK needed to inject. First, they spoof empty push-ACKs with the in-window sequence while guessing in-window ACK numbers. Once the spoofed packets trigger another challenge-ACK, an in-window ACK number is found. Finally, the attacker continually spoofs empty TCP data packets with the in-window ACK and sequence numbers as it decrements the sequence number after each send. The victim will respond with another challenge ACK once the attacker spoofs the exact sequence number minus one. The attacker can now inject arbitrary payloads into the ongoing encrypted connection using the inferred ACK and next sequence number.

… and it continues like that, but I'll spare everyone more of this.

The first parts of this so-called attack amounts to simply probing the state of the TCP/IP stack by flooding its IP and port space with packets designed to elicit a reply. Then they spoof packets from an assumed public server to do the same. But the total downfall of this is that the VPN's encryption is never penetrated or broken. They aren't seeing into any packets, nor can they meaningfully hijack any VPN communications because they don't have the connection's private encryption key... And they have absolutely no way to obtain it. They are "inferring" whether a packet is an ACK or a Reset by examining the packet's size, specifically because they cannot see into ANY of the VPN's encrypted packets.

It's true that the IP probing and scanning does create a form of information disclosure, and that VPNs are supposed to completely shield us from that. But, as we can see, it is FAR FAR from being a hijack of VPN connections. That's just not what this probing allows.