Security Now! #882 - 08-02-22

Rowhammer's Nine Lives

This week on Security Now!

This week we're going to note an urgent vulnerability created by an add-on to Atlassian's Confluence corporate workgroup server. Next week's Usenix security conference will be presenting TLS-Anvil for testing TLS libraries. Google has decided to again delay their removal of 3rd-party cookies from Chrome, and attackers were already switching away from using Office Macros before Microsoft actually did it. We have a bunch of listener feedback, some thoughts about computer science theory and bit lengths, and some interesting miscellany. Then we're going to look at the return of Rowhammer thanks to some new brilliant and clever research.

"Keptin... We cannot go to Warp"

Security News

Atlassian's “Confluence” under attack

The Australian software company Atlassian offers Confluence, a web-based corporate WIKI, written in Java and first written and published 18 years ago, back in 2004. It's one of Atlassian's most popular products, packaged as a standalone turnkey that includes a built-in Tomcat web server and HSQLDB database, though other databases can also be used. Confluence modern collaborative software. It's used to help teams collaborate and share knowledge. In a corporate setting it's used to capture project requirements, assign tasks to specific users, and manage one or more calendars. Confluence on-premise servers are widely used in corporate and government environments

We're talking about it today because Confluence appears to have recently come to the attention of prolific attackers who have been scouring it for vulnerabilities. And, unfortunately, as with any product that was first written 18 years ago, before today's hyper-attack climate existed, some code of its coding practices from back then appear are being turned into vulnerabilities.

Although Confluence has recently suffered from a series of troubles, it popped up again due to a very serious problem with the installation of default hard-coded credentials. As we know, any time you hear that a problem has been created by the discovery of default hard-coded credentials, meaning a username and password with privileges no one should have, it's difficult to chalk that up to anything other than someone's very bad decision.

In this instance, the trouble arises from the installation of an app called “Questions for Confluence” which is often installed on Confluence Server and Data Center on-premise instances to allow employees to ask questions and receive answers from a company's various internal departments. The installation of that app creates a user named "disabled system user" with the username "disabled system user" and the eMail of "dontdeletethisuser@email.com". And contrary to the name given to the account, it's not at all disabled, though it is, indeed, a system user.

Atlassian released a patch that disables this built-in hardcoded account 13 days ago, on July 20. However, Confluence server admins didn't have much time to install Atlassian's fix because the system account username and credentials for this account were posted to Twitter the next day by someone using the handle “annoyed researcher.” That's uncool.

And, as things go these days, it didn't take the bad guys long to put “annoyed researcher's” information to nefarious use. The cybersecurity firms Greynoise and Rapid7 have both reported seeing immediate and continuing exploitation of this vulnerability within days of the release of the patch. Since then, CISA has urged Confluence server owners to check and see if the vulnerable app had been installed on their servers and if so to install Atlassian's patches.

Atlassian noted that disabling the app doesn't fix the issue because it's the app's installation that created the problem. So Confluence server owners must either install the security fixes or manually disable the hard-coded “disabled system user” account created by the Questions for Confluence app.

I would recommend Confluence be placed behind a VPN or reverse proxy (eg Azure App Proxy) requiring authentication. It's simply too historically vulnerable to leave online, you're a sitting duck.

It's difficult to argue with Kevin's logic. Everyone knows that I think this should be standard practice for all publicly-exposed services: Don't publicly expose anything that doesn't utterly and absolutely, by definition, need to be publicly exposed. Web servers need to be publicly exposed by definition. So do eMail and probably DNS. The principle of exposing a minimum attack surface really means minimum. Anything that someone needs to authenticate to before receiving any services should be given an extra layer of protection.

TLS-Anvil

During next week's 31st Usenix Security Conference, being held Wednesday through Friday the 10th through the 12th in Boston, MA, a team of security researchers will be unveiling their "TLS-Anvil." The point is to use their TLS Anvil to forge fully secure and compliant TLS connection libraries. In the description of their forthcoming talk, they explain:

Although the newest versions of TLS are considered secure, flawed implementations may undermine the promised security properties. Such implementation flaws result from the TLS specifications' complexity, with exponentially many possible parameter combinations. Combinatorial Testing (CT) is a technique to tame this complexity, but it is hard to apply to TLS due to semantic dependencies between the parameters and thus leaves the developers with a major challenge referred to as the test oracle problem: Determining if the observed behavior of software is correct for a given test input.

In this work, we present TLS-Anvil, a test suite based on CT that can efficiently and systematically test parameter value combinations and overcome the oracle problem by dynamically extracting an implementation-specific input parameter model (IPM) that we constrained based on TLS specific parameter value interactions. Our approach thus carefully restricts the available input space, which in return allows us to reliably solve the oracle problem for any combination of values generated by the CT algorithm.

We evaluated TLS-Anvil with 13 well known TLS implementations, including OpenSSL, BoringSSL, and NSS. Our evaluation revealed two new exploits in MatrixSSL, five issues directly influencing the cryptographic operations of a session, as well as 15 interoperability issues, 116 problems related to incorrect alert handling, and 100 other issues across all tested libraries.

https://tls-anvil.com/assets/files/TLS-Anvil-Paper-c3dbb77c9b27783fe7998d09765061c4.pdf

The 13 libraries tested were: BearSSL, BoringSSL, Botan, GnuTLS, LibreSSL, MatrixSSL, mbed TLS, NSS (Netscape Security Suite), OpenSSL, Rustls, s2n, tlslite-ng and wolfSSL. Through the years we've talked about most if not all of these. Recall, for example, that "s2n" is Amazon's from-scratch open source implementation of TLS. They wrote it because OpenSSL which had

grown in size to half a million lines of code, 70,000 of which were tied to TLS processing, could be replaced by about 6,000 lines of code, which is what Amazon created for their AWS services.

Their paper provides a chart which cross references those 13 TLS protocol libraries with the number of exploits found, problems with the handshake cryptography, problems affecting interoperability, the generation of problem alerts, and others.

Library	Exploit	Crypto	Interop.	Alerts	Other
BearSSL	0	0	1	15	4
BoringSSL	0	0	0	6	3
Botan	0	0	0	3	3
GnuTLS	0	0	1	9	10
LibreSSL	0	1	1	7	6
MatrixSSL	2	2	7	6	16
mbed TLS	0	0	1	14	5
NSS	0	0	0	7	6
OpenSSL	0	0	0	6	7
Rustls	0	0	1	15	7
s2n	0	1	0	13	12
tlslite-ng	0	0	0	2	10
wolfSSL	1	1	3	13	11
	3	5	15	116	100

Their full 18-page paper is full of really interesting details. And they have definitely found problems of varying degrees in every one of the 13 TLS libraries. Looking over the chart, it appears that the best libraries were BoringSSL, Botan, NSS, OpenSSL, s2n and tlslite-ng.

But these guys now really understand the state of play with the industry's TLS support. They define two degrees of success for their test, writing:

> *A **strictly succeeded test** means that a library behaved exactly as expected. If multiple test cases are performed during the execution of a test template, the system under test must have behaved correctly across all of them.*
>
> *A **conceptually succeeded test** means that an implementation did not precisely fulfill the RFC requirements or did not do so in all test cases but effectively behaved correctly. This usually applies to tests where a fatal alert was expected, but the library either only closed the connection but did not send an alert, or the alert description did not match the RFC's specification.*

And then, giving some examples from their findings:

> *We count all tests as 'passed' that either succeeded strictly or conceptually and include the percentage of passed tests. We further list the ratio of conceptually to strictly succeeded tests as an additional metric to compare how close an implementation is to the RFC for our tests.*
>
> *Rustls, for example, passed many tests, but most of them only succeeded conceptually as*

Rustls often did not send any alerts. Botan, in contrast, often fulfilled the expectations of our tests and, at the same time, was very accurate with alert descriptions. We generally found that most libraries pass a high ratio of the test templates, with NSS, BoringSSL, tlslite-ng, and OpenSSL passing around 97% of their applied server tests. Among the client tests, BearSSL, BoringSSL, and Botan have the highest ratio of passed tests with 97.3%, 96.8%, and 96.2%, respectively.

We expand upon the results of libraries with significantly worse ratios, listing how many test templates of an RFC passed and how many were executed for each library. We grouped the results of test templates based on similar error cases and identified a total of 239 issues. We further categorized these findings based on their impact and determined that three immediately led to exploits in wolfSSL and MatrixSSL.

Additionally, we found five issues affecting the cryptography of a handshake. As an example, the clients of MatrixSSL, s2n, and wolfSSL are willing to negotiate parameters they did not offer. While none of the parameters negotiated are (sufficiently) weakening the security to pose an immediate threat now, parameter negotiation is a basic security property of every cryptographic protocol to prevent current and potential future attacks.

We identified 15 issues affecting the interoperability to an extent where a peer that operates within the boundaries of the RFC may not be able to complete a handshake. Note that this may also include intentional deviations by the developers if they break the implementation's correctness in regards to the specification.

100 issues account for various likely uncritical cases where a library deviated from the RFC beyond alert codes and where interoperability should not be affected. Examples of these findings are a bug in OpenSSL, which allowed multiple TLS 1.3 HelloRetryRequest messages, which can keep the client in a handshake loop, or the support of deprecated curves by mbedTLS.

Finally, we grouped 116 cases where a library did not send an alert or sent a different alert than requested by the RFC. These are minor deviations from the standard. However, in the past, information gained from the type of alert sent by an implementation has been used to mount side channel attacks. To avoid such deviations, great care must be taken when designing the alert handling of an implementation. We hence chose to include these findings in our reports to the developers.

And what they had to say about responsible disclosure and feedback was very interesting:

We responsibly disclosed all of our findings to the respective developers. During the disclosure process, multiple developers stated that they intentionally violate RFC requirements in specific cases. As an example, in TLS 1.2 peers are not allowed to resume a session that has been terminated by a fatal alert. However, when multiple sessions take place in parallel, this requirement is difficult to implement. Multiple developers also stated that they intentionally send different alerts or no alerts at all. One reason was to minimize the risk of creating an alert-oracle for attacks. We do, however, stress that the specified alert handling of current TLS RFCs does not result in a known, exploitable oracle but is considered to be secure. Our original test suite contained 18 additional test templates, which we removed from the test suite after discussions with different library developers. Their reasoning convinced us that in these cases our interpretation of the RFC was too strict and that their library behavior was indeed valid.

Our presented evaluation does not contain these additional test templates anymore.

There were also some cases where the developers argued that it is unreasonable to follow the specification. For example, in some tests, a server that supports TLS 1.3 and TLS 1.2 would need different alert handling for the same effective test. The situation becomes even more tricky when the server has not decided which protocol version to speak yet. For example, a server that receives a malformed or illegal ClientHello message would first need to evaluate the supported protocol version of the client to decide upon the correct alert handling rules. Correctly handling these nuances can be very complex, and it is arguable whether the strict RFC conformance across all supported versions is worth the added complexity.

The security bugs we reported and most of our other reports have been acknowledged by the developers and will be considered for future releases. Since most failed test templates only failed for single or very few libraries, we conclude that the developers in general share our understanding of the RFCs.

So, another way to read this is: Ivory tower RFCs meet real world implementations. There are some places where it just doesn't make sufficient sense to go to all of the trouble that absolute slavish adherence to the RFC would require.

All of this beautiful work is public on Github, and the site: https://tls-anvil.com/ offers the Dockers to allow the use of these tests by anyone. This is precisely the sort of work we need more of.

Google delays Chrome's cookie phase-out again

Last Wednesday, Google's vice president of their Privacy Sandbox initiative, Anthony Chavez said *"The most consistent feedback we've received is the need for more time to evaluate and test the new Privacy Sandbox technologies before deprecating third-party cookies in Chrome. This feedback aligns with our commitment to the CMA to ensure that the Privacy Sandbox provides effective, privacy-preserving technologies and the industry has sufficient time to adopt these new solutions."*

Okay. So let's back up a bit. Google is once again delaying its plan to terminate Chrome's support for 3rd-party cookies. Google first unveiled its Privacy Sandbox initiative back in 2019 with the announcement that its implementation would begin **this** year, in 2022. However, last year, after being scrutinized by the UK's Competition and Markets Authority (CMA) and the US Department of Justice, Google announced their intention to delay their 3rd-party cookie phaseout until mid-2023, so next year. And now Google announced another delay, saying that it won't end support for 3rd-party cookies before the second half of 2024.

Under its current timeline, Google will expand the availability of its Privacy Sandbox trial to "millions of users globally" by early next month. The company then plans to gradually roll out the test to more individuals throughout 2022 and 2023. It hopes to officially launch the Privacy Sandbox APIs by the third quarter of 2023. Anthony said: *"This deliberate approach to transitioning from third-party cookies ensures that the web can continue to thrive, without relying on cross-site tracking identifiers or covert techniques like fingerprinting."*

We've been tracking Google's proposed technologies closely. Their first attempt at replacing 3rd-party cookies was their awkwardly named "Federated Learning of Cohorts" or FLoC. Then, when that failed to liftoff — largely because no one understood how it worked and it was too opaque — Google announced their new approach called Topics, which is largely the same but is far more understandable and transparent. As the name suggests, Topics more transparently works to track our interests by site... under approximately 300 different topics. When we visit a website that requests the information, Chrome will offer three topics we're interested in. The site's publisher can then further share that information with their advertising partners to decide what ads to show. In theory, that should create a more private browsing experience.

Being a technology hound, I understand and love Google's initiatives. But the big question will be whether any of this can actually replace all other side-channel tracking. I think that the only way we're ever going to be free of tracking altogether is through legislation which makes it flatly and permanently illegal. We'll first need a privacy-protecting replacement, which Google's Topics does offer. But once that's in place, we're going to need governmental regulation, probably led by the EU, to completely outlaw any and all other forms of anonymous Internet user identification.

This has all been very slow and painful. But if we've seen anything it's that these sorts of changes take time, are always slow and often painful. But it's clear that progress is being made.

Attacker responding to loss of Office Macros

ProofPoint released the results of their eMail-based malware attack analysis. In their report they observed that their analysis of campaigned threats, which include threats manually analyzed and contextualized by Proofpoint threat researchers, the use of macro-enabled attachments by threat actors decreased approximately 66% between October 2021 and June 2022.

Remember that Microsoft previously announced it would begin to block XL4 and VBA macros by default for Office users in October 2021 for XL4 (which coincides exactly with the beginning of the observed drop of their use) and also in February of this year for VBA. And as we know there was some confusion last month, but the disabling has ultimately remained in place, thank goodness.

So, threat actors across the landscape have responded by shifting away from macro-based threats. Based on Proofpoint's campaign data since October 2021, threat actors have pivoted away from using macro-enabled documents attached directly to messages to deliver malware, and have increasingly used container files such as ISO and RAR attachments and Windows Shortcut (LNK) files.

Microsoft blocks VBA macros based on the so-called "Mark of the Web" (MOTW) attribute which tags files based upon their source. The Mark of the Web identifies whether a file originated from the Internet Zone. The Mark of the Web (I just love saying that) tags the file that's downloaded. But it cannot tag files contained within other files. Therefore, the Mark of the Web can be bypassed by using container file formats. An IT security company Outflank has detailed multiple options for red teamers to bypass MOTW mechanisms, and of course these techniques can be used by threat actors as well.

Threat actors can use container file formats such as ISO (.iso), RAR (.rar), ZIP (.zip), and IMG (.img) files to sneak macro-enabled documents into a user's machine. The container gets flagged but its contents do not so the file system will not identify the document as coming from the web and some of these new defenses will be bypassed.

So the cat and mouse race continues. But having Microsoft finally hampering the auto running of these macros by default was still a huge step forward. I'm still somewhat amazed by the amount of pushback Microsoft received. People want to have the power without being willing to take responsibility for what having it means. So Microsoft finally, after decades, did the right thing by simply taking that power away while providing them with ample safe workarounds.

SpinRite

Lewis in the spinrite.dev newsgroup:

> *I was surprised to hear this week that Spinrite 6.1 is still using 16 bit addressing, limiting it to 64K (erm, I forgot the word used). Is this actually helpful in 2022 and wouldn't it be simpler for the code to use 32 bit addresses rather than swap between 64K blocks?*
>
> *Haven't we been at 32 bits for almost 40 years?*

Lewis' question raises an interesting point about practical computing and computer science that I wanted to take a moment to address. So let's delve for a couple of minutes into some theoretical aspects of computer science which not everyone may have stopped to consider...

First of all, let's understand that whatever they are being used for, the number of binary bits collected together determines the number of possible states that the collection of bits can have. We might be using a collection of bits to represent a value, in which case the number of bits determines the number of possible discrete values that can be represented by that collection. Or the collection of bits might be a cryptographic key, in which case the number of bits in the collection determines how many wrong keys that one right key is hiding among. Or the collection of bits might be a pointer on one object in a linear array of other objects. In that case the number of bits in the collection determines how many discrete objects that collection of bits is able to refer to.

So far, it would appear that the more bits the merrier. After all, what's wrong with being able to represent more values than you might need to? Or what's wrong with being able to refer to more objects than you have? In all cases, logic would suggest that there will be some minimum number of bits required to do the job, but that having unused bits would not be a problem.

There are at least two problems with having unneeded and unused bits. One is power consumption. All other things being equal, and they pretty much are, 32 bits consumes four times the power of 8 bits. And those 32 bits consume equal amounts of power whether they are 0's or 1's because they still need to be checked and moved around.

And speaking of moving around, whatever the bit size is, all of those bits need to be moved around. They probably need to be loaded and stored from and to main memory. But as we know, memory bandwidth has become one of the limiting and crippling factors in modern computing. Since main memory has been unable to keep up with our data-hungry processors, all manner of multi-level caching has been deployed in an attempt to decouple the slow main memory from our super-fast super-hungry CPUs. So, if a CPU is using 32 bits to refer to a small collection of objects that could be referred to with only 8 bits, then 24 bits out of 32 bits of precious memory bandwidth are being completely wasted. The CPU is only obtaining 25% of the system's possible memory bandwidth if, as in this example, 32 bits are being used where 8 bits would have been sufficient. The other consequence of unused bits is the size of the program's code and data. If the code is using 32 bits where 8 would have been sufficient, the resulting code will be much larger, by as much as four times. That means it takes as much as four times more space, takes four times longer to load, and runs as much as one quarter the speed since there's just so much more code bulk for the CPU to slog through. So, this all tells us that the theoretical optimal number of bits to use would be just as many as are required, but no more.

There were, back in the early days of digital computer design, variable bit length computers. Early serial memories such as paper and magnetic tape, drum and delay-line did not inherently bring along any natural boundaries, so they worked well with the idea of variable length computing. And remember that back in the 1940's and 50's this was all being invented. No one back then knew what the answer would be. So perhaps variable length computing was the best.

But when core memory was invented, it was far more efficient to "stack" the "planes" of core memory grids and to read out a fixed number of bits, in parallel, one from each memory plane. So it became natural for computers to load and store data in the same bit lengths as the core memory could produce or consume per cycle. And that approach of grouping data bits into bytes and words having fixed lengths has stuck with us ever since.

So what does all this have to do with SpinRite?

SpinRite was born when the IBM PC, using the 8088 x86 chip, had 16-bit registers and 20-bits of addressing. 16-bit registers could represent 64K different values. And 20-bits of addressing could refer to one megabyte of RAM. But back then, one megabyte was so much memory that it wasn't even standard. It was quite common for the first PCs to have 256K or 512K of RAM, because programs back then were small, efficient, and didn't really do very much.

The PC's BIOS, MS-DOS and all of the DOS clones were, and still are today, 16-bit code and they only run 16-bit programs. Yet an incredible amount of work was accomplished by WordStar, WordPerfect, VisiCalc, Lotus 123, dBase III, FoxPro and many other 16-bit programs ... including SpinRite. Today's SpinRite is a 16-bit program because it's running on the 16-bit DOS operating system. And even today, I'm not unhappy about that because most of what SpinRite is doing fits nicely into 16 bits.

But even back then, many of those other programs were bursting at their seams within the constraints of 16-bit code. The PC's memory was first Expanded and then Extended using elaborate page-swapping schemes to allow data hungry tools like Lotus 123 to operate upon much larger data sets. And toward the end of all that, DOS extenders were created to host true

32-bit programs on 16-bit DOS. Doing this was an act of sheer desperation. It was the definition of "kludge", because DOS could only run under the CPU's "real" mode and "protected mode" was required to go beyond 16 bits. So the CPU was dynamically switched back and forth between real mode and protected mode on the fly. What a mess.

But SpinRite is not pure 16-bit code. It's true that it's running in real mode alongside the 16-bit BIOS and 16-bit DOS. But SpinRite v6.1 takes advantage of what must have been an inadvertent bug in the very first 80286 chips and which, after appearing in those chips, Intel must have been worried about breaking backward compatibility if the bug was fixed. The bug arises from the fact that the limitations inherent in the 8088's and 8086's real mode can be simulated by protected mode. So the 80286 chips and all of the chips that have followed since don't actually have real mode. They have what amounts to a "clamped down" protected mode with full access to the system's I/O hardware. No restrictions there.

Since I've talked about this before, I won't go into detail about it again. But suffice to say that SpinRite uses this bug to obtain 32-bit access to any machine's first 4 gigabytes of RAM from within 16-bit real mode code. It's kinda the best of both worlds. SpinRite itself remains small with most of it remaining 16-bit code. But when and as necessary, it's able to directly address any memory that's accessible with 32-bits, which is 4 gigabytes. Doing this was necessary in order to obtain the absolute maximum possible performance from AHCI-connected drives by reading very large 32K-sector, 16-megabyte blocks of data at a time.

But in real mode, the program counter which points to instructions is still 16 bits long. So code must be executed within 64K-byte blocks. And one of those is what SpinRite has outgrown, thus now needing a second code segment block to contain its overflow code.

Once SpinRite v6.1 is finished and published, I plan to immediately begin the work of re-coding 16-bit SpinRite into pure 32-bit code. It's time to finally make that move. The result will be a larger SpinRite, since, as I've noted, for most of what SpinRite needs to do, 32 bits is excessive and unused bits are truly wasteful. But it's also a lot more convenient and fun to code without any limitations, even if it is technically less efficient. At least, for the foreseeable future, it'll still be Intel assembly language!

Closing The Loop

Jonathan Leitschuh tweeted this yesterday and then followed up with eMail to TWiT to be sure that we had seen it. So I wanted to let him know that we had. He tweeted:

Jonathan Leitschuh @ Open Source Summit / @JLLeitschuh

> *@SGgrc @leolaporte I've been listening to @SecurityNow for years. This year I'm speaking at @BlackHatEvents, @defcon, and @BSidesLV**!** I learned so much from your show. I wouldn't be where I am today without it. Thank you so much for the amazing education you've provided!* ❤ [he continued in his next tweet] *@SGgrc it's from your stories of great hackers like @taviso* [Tavis Ormandy] *(and Google Project Zero), @dakami* [Dan Kaminski]*, and many others, that I learned the norms of vulnerability disclosure. I learned the importance of building software that is secure-by-default.*

Alex Neihaus / ☁️ @yobyot

I esp. enjoyed your and Leo's mirth at the astonishing flaws in the MV720 in episode 881. I worked in China in the mid 90s. This doesn't surprise me. It's a cultural thing — a fundamental business drive to maximize profit by cheapening the product.

mark0j / @mark0j5

Hi Steve, long time listener here. Love your show. Just wanted to drop an information that there is an open source version of Tailscale and is called Headscale. Seems good alternative to the paid version. Available here: https://github.com/juanfont/headscale

Henning @ihbrune

Dear Steve, just listened to GRC 881 and your criticism of the way Microsoft implemented the protection against brute force password guesses (allow 10 failed logins, then ban any logins for 10 minutes, then allow 10 failed logins again and so on...).

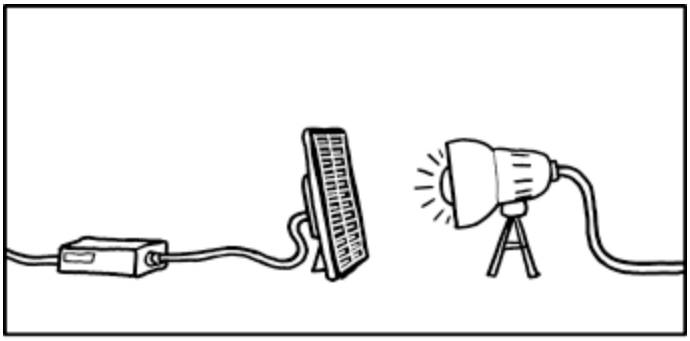
You asked why Microsoft did not block the logins indefinitely. And this is my answer, because I designed a login system for our universities web applications, that works the same way:

If we would block accounts then it would be easy for anyone of our aspiring computer science students to produce a small script that locks out all of our users in one night. And again next night. And again....

This is a trade off between protecting the passwords of our users from brute force attacks and thereby creating the risc of opening a door for denial of service attacks.

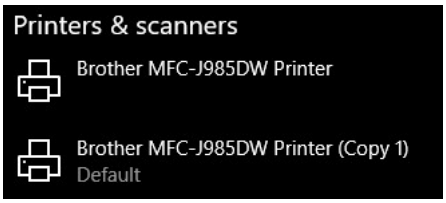
I think one password guess per minute is slow enough if you don't allow completely trivial passwords in your userbase to protect you services.

From many of our XKCD-following listeners:



ENERGY TIP: INCREASE THE SECURITY OF YOUR HOME POWER SUPPLY BY INSTALLING AN AIR GAP.

"📡 Richard COVID 💉💉💉💉 Flu💉 😷" / @pa28



Hiveware / @rtischer8277

> *@SGgrc A little pushback on your Windows OS being creaky and brittle. Take a look at 11's Settings controls. No longer are they a thin veneer over ancient code. They are a beautiful evolution of the venerable PropertySheet ctrl. Top to bottom. Credit to MS where credit is due.*

After reading this tweet and wanting to include it in the podcast for the sake of balance, I thought that I might have missed something. So I fired up a VirtualBox VM where I have an instance of Windows 11 and I went over to the Settings controls. First of all, this looks exactly like Windows 10. Perhaps I missed what Hiveware was referring to, but I poked around and opened a bunch of control sub-panels. What's there appears to be a mixture of old stuff and new stuff. It's easy to find a sub-dialog that's been around since Windows 98 and 2000. A perfect example is the control panel for Sound. It looks exactly like what we had, perhaps as far back as Windows 3.1. Or the "Internet Options" dialog, or "Device Manager." I could go on and on. Hiveware said "*No longer are they a thin veneer over ancient code.*" but I think that's precisely what those ancient dialogs are. They **are** ancient code, and I think that's great because those are the parts of Windows that work most reliably.

But I also agree with Hiveware's statement that the new style is beautiful. It's just that Microsoft didn't change any of the older dialogs so that everything matches and looks the same. So we end up with several decades worth of user interface ideas strewn throughout the user's experience. And this is all just UI frosting. None of which speaks to whether or not the underlying Windows OS being creaky and brittle.

In the Win11 VM, I just received a notice about whether I was ready to change my backup options. When I clicked the "not now" button, the dialog shrank back, as if I had pushed it back from the screen's surface, then exited stage right. It certainly is pretty. If only it could print.

Miscellany

RIP: Nichelle Nichols

Just a note that this past weekend having reached age 89, the world lost Lieutenant Uhura from the original Star Trek series. Nichelle Nichols brought us Uhura throughout the original series as well as all six subsequent feature length Star Trek films. The bridge wouldn't have been the same without her.

We've previously lost Spock, Doctor Leonard "Bones" McCoy, Scotty and Nurse Chapel. Meanwhile the actors who gave us "James Tiberius Kirk", "Pavel Chekov" and "Hikaru Sulu" remain with us.

What an incredible legacy they all had a key role in creating. And some great stories they brought us.

"The Dropout" on Hulu and **"WeCrashed"** on AppleTV+
I really enjoyed both series. For those who don't know the Theranos story, a female Stanford University undergrad, who wanted to achieve Steve Job's style success and who believed that all she needed was to want it badly enough, decided that being able to perform hundreds of blood screening tests given a single "pin prick" drop of blood would change the world. In that, she was right. That would certainly have changed the world. But she was wrong in her belief that all she needed to do was want it badly enough. Believe it or not, she had no education or training in the science that realizing her dream depended upon. None whatsoever. There was never any reason to believe that it was possible and there was a lot of reason to think that it wasn't. But those pesky details didn't even give her pause.

Being a bit of a biohacker myself, I was interested enough in the blood testing technology that Elizabeth Holmes and Theranos were attempting to commercialize that I followed-up watching that series which I found fascinating with some YouTube and the US patent and trademark office research. I was curious to get some sense for how accurate "The Dropout"'s portrayal of those events and its characters was. For anyone else who might be interested or curious, given the available public record, everything portrayed in "The Dropout" was astonishingly accurate, right down to the lighting used to shine off the corneas of Elizabeth Holmes' eyes in a series of memorable close ups. I even found a multi-page document which dissected the "Edison" mobile lab machine that they were never able to get to work. What was clear was that even if it had worked, it would never have been able to achieve what they had been promising.

Winamp releases new version after four years in development
I don't recall what caused me to mention WinAmp last. But I did, adding something like "Really?" and I was immediately scolded by a number of our listeners who said: *"Hey, I'm still using it! It's the best media player ever created by man!"* I can certainly respect that since I've created a few things that others regard as "the best ever", even though they've been around for quite a while.

So I just wanted to give our intrepid die hard WinAmp fans the news that after four years of development, the first new release candidate of WinAmp has been released. Lawrence Abrams over at BleepingComputer was where I learned of this. And what he had to say was, I thought, interesting. Lawrence wrote:

> *Winamp has released its first release candidate after four years in development, officially bringing the popular media player out of beta. Before music streaming platforms rose to prominence, we needed to rip our music from CDs and play the resulting MP3 files on a media player. One of the most beloved media players to play MP3s was Winamp, which included retro skins and animated visualizations that synced with your music.*
>
> *Winamp ceased development after version 5.666 was released in 2013. That was until October 2018, when Winamp 5.8 was leaked online, and the developers decided to publish it*

themselves on the Winamp.com website. [I think that leak in 2018 must have been the reason I mentioned it previously.]

Since then, the developers have promised an updated version with cloud streaming support and more modern features. Finally, in November 2021, the Winamp.com website received a facelift with a new logo and a beta signup form to be notified when new versions were released.

Last week, Winamp 5.9 RC1 Build 9999 was released, marking it as the first version released in 4 years and as the first release candidate of the revitalized media player. While the Winamp release candidate does not contain too many changes, the main goal of this release was to upgrade the code base from Visual Studio 2008 to Visual Studio 2019. Now that this has been completed, the team can add new features and capabilities to the media player.

A byproduct of these changes is that Winamp requires Windows 7 SP1 or later, dropping support for Windows XP and Vista.

The WinAmp team wrote:

"This is the culmination of 4 years' work since the 5.8 release. Two dev teams, and a pandemic-induced hiatus period in between," reads the changelog for Winamp 5.9 RC1 Build 9999.

"To the end-user, it might not seem like there's a whole heap of changes, but the largest and hardest part was actually migrating the entire project from VS2008 to VS2019 and getting it all to build successfully.

"The groundwork has now been laid, and now we can concentrate more on features. Whether fixing/replacing old ones or adding new."

Rowhammer's Nine Lives

https://andreaskogler.com/papers/halfdouble.pdf

Yes. Rowhammer is back, alive and kicking. And it really shouldn't surprise any of us by now. The principle we've seen and learned over and over in context after context is that "mitigating" a security flaw is not the same as fixing it — and security flaws turn out to be stubborn things. All we need to do is look at the continuing saga being brought to us by Spectre and Meltdown, where the discovery of fundamentally exploitable flaws arising from the advanced performance optimizations which have been deeply built into modern CPUs, which can be used to leak secrets across process isolation boundaries, to see an example of a family of flaws that refuse to die.

And as we know, another completely unrelated, yet still fundamental flaw, was discovered in the operation of today's dynamic random access memory. Security researchers discovered that modern memories had become so hyper-dense with every possible margin engineered out, that adjacent rows of bits which were, of course, supposed to be completely independent of one another, were often able to interfere with each other. And being the super-clever engineers that they are, these researchers figured out how to turn the resulting spontaneous "bit flips" that could be induced to occur into active and effective security exploits.

The DRAM manufacturing industry responded, not by reducing the storage density of RAM back down to where there would be sufficient noise immunity to prevent adjacent row interference. No. It responded with a — wait for it — "mitigation" of the problem; by making the memory system still more complex by selectively refreshing the endangered rows of DRAM which surround the rows which might be causing interference, whether that interference was deliberate or accidental. This was named "Target Row Refresh" or TRR.

You know you're in trouble when DRAM is the subject of an apparently endless series of CVE-worthy exploits. There was the original RowHammer. Then we had "DRAMmer" (double Rowhammer), "RAMBleed", "SPOILER", "TRRespass" and last year's "Blacksmith". And now, having not yet used up nine lives, we have the "Half-Double" attack. The name for this latest attack comes from a crochet stitch which is taller than a single but shorter than a double.

The point is, the actual fundamental problem of DRAM activity noise immunity being too low was never addressed. In case after case, since the cost of treating the underlying disease was too high — it would have meant backing off on DRAM storage density — instead of treating the disease, one symptom after the next has been treated. So today, even the most recently engineered DRAM has remained diseased with some fancy patchwork added in an attempt to shore up one of its fundamental problems.

I named today's podcast "Rowhammer's Nine Lives" because this problem refuses to die. These amazing researchers are back, having — almost not surprisingly — worked around the most recent iteration of Rowhammer mitigations.

Their paper, which will be delivered during the same USENIX security conference next week where TLS-Anvil will be shown, is titled *"Half-Double: Hammering From the Next Row Over"* The Abstract of their paper explains:

Rowhammer is a vulnerability in modern DRAM where repeated accesses to one row (the aggressor) give off electrical disturbance whose cumulative effect flips the bits in an adjacent row (the victim). Consequently, Rowhammer defenses presuppose the adjacency of aggressor-victim pairs, including those in LPDDR4 and DDR4, most notably TRR.

In this paper, we present Half-Double, an escalation of Rowhammer to rows beyond immediate neighbors. Using Half-Double, we induce errors in a victim by combining many accesses to a distance-2 row with just a few to a distance-1 row. Our experiments show that the cumulative effect of these leads to a sufficient electrical disturbance in the victim row, inducing bit flips. We demonstrate the practical relevance of Half-Double in a proof-of-concept attack on a fully up-to-date system. We use side channels, a new technique called BlindHammering, a new spraying technique, and a Spectre attack in our end-to-end Half-Double Attack. On recent Chromebooks with ECC- and TRR-protected LPDDR4x memory, the attack takes less than 45 minutes on average.

What these guys have figured out is sheer brilliance. They came up with a way of turning the Target Row Refresh Rowhammer mitigation, which is now present in all of the latest DDR4 DRAM, against itself. They've figured out how to use TRR to **induce** the problem that it's designed to prevent.

Here's how they describe what they're accomplished:

Rowhammer is a widespread DRAM issue caused by the unintended coupling between its constituent rows. By repeatedly accessing one row (the aggressor), an attacker can corrupt data in adjacent rows (the victims) by accelerating their charge leakage. As a powerful means of bypassing hardware and software memory protection, Rowhammer has been used as the basis for many different attacks.

*Previously, Rowhammer was understood to operate at a distance of **one** row: an aggressor could only flip bits in its two immediate neighbors, one on each side. This makes intuitive sense: as a coupling phenomenon, the Rowhammer effect should be the strongest at closest proximity. Indeed, this assumption underpins many countermeasures that have been proposed against Rowhammer, especially the ones that rely on detecting aggressors and refreshing the charge in their intended victims. In fact, Target Row Refresh (TRR), a productionized countermeasure widely deployed as part of Low Power DDR4 or standard DDR4 chips, falls into this detect-and-refresh category.*

In this paper, we present Half-Double, a new escalation of Rowhammer where we show its effect to extend beyond just the immediate neighbors. Using Half-Double, we are able to flip bits in the victim by combining many accesses to a far aggressor (at a distance of two) with just a few to a near aggressor (at a distance of one). Both aggressor distances are necessary: accessing just the far aggressor does not flip bits in a row that's two away, whereas accessing just the near aggressor devolves into a classic attack that's easily mitigated. Based on our experiments **[get a load of this!]***, the **near** aggressor appears to act as a bridge, transporting the Rowhammer effect of the far aggressor onto the victim. Concerningly, TRR actually **facilitates** Half-Double through its mitigative refreshes, turning their recipient row into the near aggressor that co-conspires with the far one that necessitated the refresh in the first place. In effect, the cure becomes the disease.*

While the discovery and evaluation of Half-Double is the main contribution of this work, we also demonstrate its practical relevance in a proof-of-concept exploit. However, current systems limit the attacker's control, introducing 4 challenges:

First: The adversary needs to allocate memory contiguous in a DRAM bank. However, without access to physical addresses and huge pages, we have to introduce a novel approach combining buddy allocator information with a DRAM timing side channel to reliably detect contiguous memory.

Second: ECC-protected memory can make bit flips unobservable depending on the victim data which the attacker does not control, the adversary cannot template the memory like in previous Rowhammer attacks, since hammering requires knowledge of the cell data. As the state-of-the-art does not solve this problem, we introduce a novel technique called Blind-Hammering to induce bit flips despite the ECC mechanism of LPDDR4x.

Third: Reduced address space sizes on recent ARM-based systems break the page table spraying mechanism from previous attacks. Therefore, we develop a new spraying technique that is still unmitigated.

Finally, fourth: Without templating, we need an oracle telling whether Rowhammer induced an exploitable bit flip, without crashing the exploit. For this, we introduce a novel approach using a Spectre-based oracle for exploitable bit flips.

[And here it is...] *We combine these techniques into an end-to-end fully-successful proof-of-concept,* ***the Half-Double Attack****, which escalates an unprivileged attacker to arbitrary system memory read and write access, in other words: full kernel privileges.*

The Half-Double Attack runs within 45 minutes on a fully updated Chromebook with TRR-protected Low Power DDR4 memory.

To summarize, we make the following contributions:

1. *We discover a new Rowhammer effect: Half-Double, and evaluate a set of devices and modules for susceptibility.*

2. *We perform a thorough root-cause analysis to empirically prove that TRR is* ***responsible*** *for the Half-Double effect.*

3. *We analyze the stop-gap mitigations present in today's systems and show that with a new exploit using Half-Double, we can bypass them and build an end-to-end attack.*

4. *Our end-to-end Half-Double Attack runs on up-to-date Chromebooks and combines the Half-Double effect with exploit techniques, side channels, and a Spectre attack.*

The moral of our story is: Responding to fundamental exploitable design flaws with mitigations only delays the inevitable. I expect that DDR5 DRAM will learn the lessons of the Half-Double attack to render it also ineffective. Will that be the end of it?