RAPID7

RESEARCH

UNDER - THE - HOODIE 2020

By Rapid7 Global Consulting

TABLE OF CONTENTS

Executive Summary

Penetration testing—the practice of simulating a criminal breach of a sensitive area in order to uncover and fix defensive failures—is (still) a rather occult subject, as evidenced by the recent dust-up with local authorities in Dallas County, Iowa. Employees from Coalfire Systems, a well-respected penetration testing firm, were arrested in September 2019 due to a misunderstanding of the scope and fundamental legitimacy of penetration testing.[1] Although charges were eventually dropped in February 2020, the incident rocked the pentesting space. Clearly, everyone involved in offensive security needs to strive to better explain the value of routine pentesting of our physical and virtual world.

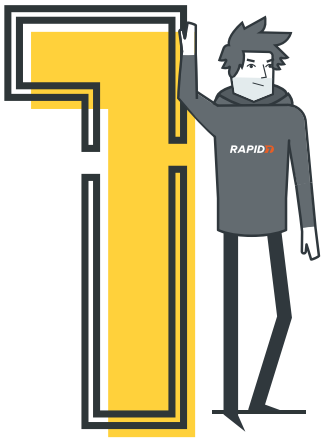
This report aims to do just that by exploring the hows and whys of penetration testing, covering mainly internal and external network compromises, with some supplementary data on social engineering and red team simulations. Over the course of 12 months worth of survey data, we found:

- Internal network configuration and patch management continue to provide "easy" soft targets to penetration testers, who can often use off-the-shelf commodity attacks to escalate privileges and move laterally about the network without being detected.
- Password management and secondary controls such as two-factor authentication (2FA) on the enterprise level are severely lacking, leading to "easy" compromises involving both password spraying and decrypting hashed passwords acquired during simulated breaches.
- As the world's knowledge workers depend more than ever on VPNs and internet-based applications, rather than onsite, traditionally internal network controls, penetration testers are finding significant flaws in those VPN terminators and custom web apps.

While none of this is particularly shocking to even the most Pollyanna security researcher (we are a cynical bunch), this is solid data that can help enterprises around the world understand what to expect from their next penetration test and be used as a checklist of what to investigate and remediate before then.

Finally, the survey data for this report was collected from June 2019 through June 2020 and collected responses from 206 engagements total. The busiest individual pen tester in the world tends to be involved in fewer than 25 penetration tests a year (assuming an average engagement time of about eight to 10 days), so this paper should be illuminating for any pen tester interested in industry-wide trends.

[1] https://www.theregister.com/2020/08/05/coalfire_pentest_iowa_black_hat/



Scope, Planning, and Execution

Before diving into the results of this year's Under the Hoodie survey, we will first define some terms. After all, the purpose of this paper is to demystify penetration testing, so let's get to some jargon busting.

Code Review

Code reviews tend to be the least "in-production" style of an engagement. They tend to focus on just one application or product, to which the penetration tester has full or nearly full access to. The goal of this engagement is most like an aggressive quality assurance review of the system or device under test, with the penetration tester's goal being to find unexpected, security-relevant bugs in that system or device.

Internal Network Assessment

Internal network assessments are comprehensive explorations of the client's internal networks. This tends to include the types of systems you'd expect to see "locally" on the LAN: Windows clients and servers, printers, internal applications, and Wi-Fi access points. Often, the tester is invited onsite or is given access to an internal access point through a VPN connection. The ultimate goal of an internal network assessment is to document and demonstrate information security risks inherent in the internal network that might be compromised by either a motivated, malicious insider, or exploited by an attacker who has achieved an internal network presence through some other means. This, of course, implies that vulnerabilities might be exploited externally, which brings us to…

External Network Assessment

External assessments, like internal assessments, seek out vulnerabilities and misconfigurations in the client's IT infrastructure, but, as the name suggests, only really those that are visible and reachable to an externally based attacker. So, these engagements tend to feature a lot of web application hacking, as well as the mapping and exploitation of services that are exposed to the internet at large: VPN endpoints, email, and any services that are exposed to the outside world by accident (Telnet consoles in my DMZ? It's more likely than you'd think!).

Social Engineering Engagements

Social engineering engagements almost always include some email component, since that is still the favored method for extracting password information (often through something like a fake login page the targeted victim is directed to) or dropping initial malware on the victim's computer that grants an external attacker insider access to the organization. More occasionally, consultants are tasked with a physical social engineering engagement, where the task at hand is to slip into a restricted or confidential area of the client organization's physical infrastructure—think locked networking closets, restricted areas of a manufacturing floor, or something of that ilk.

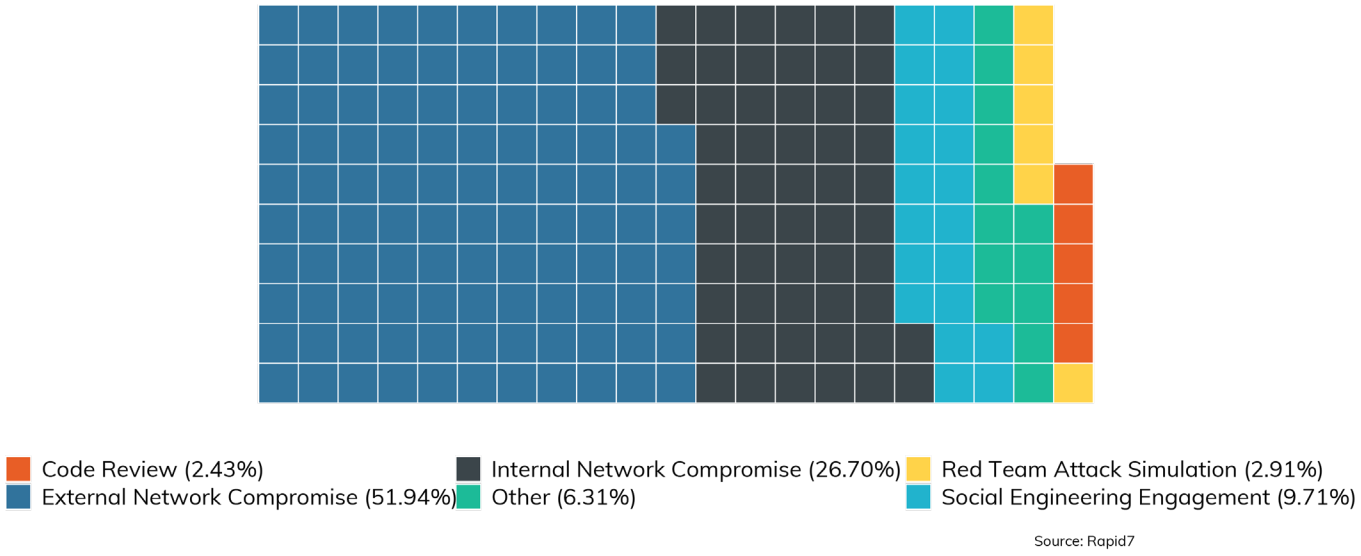
Red Team Simulations

Red team exercises are fairly elaborate affairs and tend to combine elements of a social engineering exercise (to gain the trust of a key insider), an external network assessment (to map out and exploit the internet-exposed exposures and vulnerabilities of the network), and an internal network assessment (to escalate privileges from that of an external stranger to an internal superuser). All of this happens while the client's normal detection and defense technologies are active. If a network assessment could be thought of as a 150-point automobile inspection, assessing the system as a whole, a red team engagement is more like a crash test; every system is put to the test in an exercise that most closely resembles a real-world attack.

Engagement Types, Compared

With these distinctions between engagement types defined, we can now take a look at the more broad categories of "inside" jobs versus "outside" jobs, and how often clients request each kind. Figure 1 shows a breakdown by engagement type over the surveyed set of engagements.

Figure 1: Penetration Testing Assessments Performed



Source: Rapid7

You can think of these broadly as falling into two buckets: "inside jobs" and "outside jobs." With inside jobs, the attacker already has a leg up by having access to the internal network or source code, or physical access to a device under test. By combining **Code Review** (2.43%) and **Internal Network** (26.70%), we can see that 29.13% of all engagements are inside jobs. Now, this might sound unfair, but organizations are wise to test their defensive capabilities against both the "disgruntled employee" style of inside attacker, as well as the attacker who has crossed from the "outside" to the "inside," through whatever means.

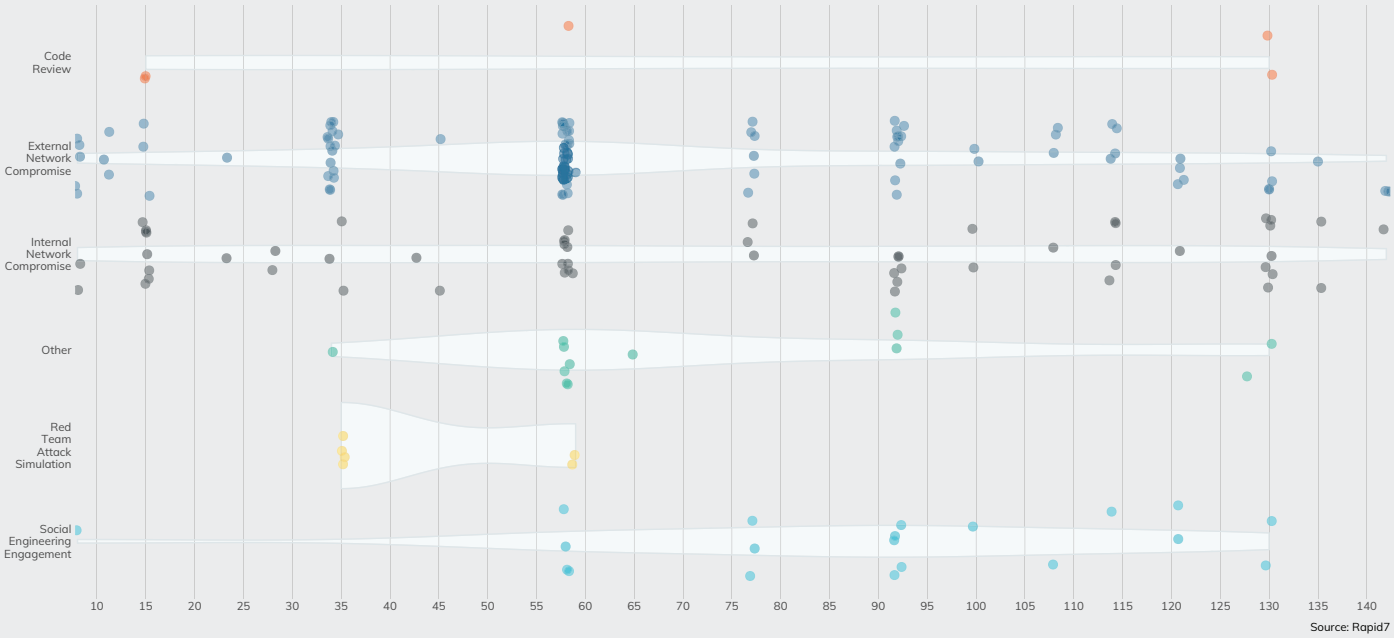
"Outside Jobs" are where the traditional external attacker operates. **External Network** (51.94%), **Red Team** (2.91%), and **Social Engineering** (9.71%) all tend to be from the "outside" and total up to 64.56%. What does this tell us? Well, pen test clients tend to be more interested in external threats rather than internal ones. In fact, they prefer externally based assessments more this year than last year—last year's "outside" bucket totaled up to 50% of all engagements, with "inside" accounting for 39.4% (in both years, "Other" was unchanged at about 8%). Since our data covers the period of worldwide lockdown due to pandemic concerns, this bump in favoring external to internal this year over last is entirely expected.

Timing Considerations

So far, these Under the Hoodie reports capture only those penetration tests that were performed by Rapid7 or Rapid7-hired subcontractors. Because of this lack of diversity in penetration testing providers (n=1), we should expect that the number of hours contracted to be fairly standardized. In the past, we saw exactly this—engagements tended to land right on the 80-hour mark for the number of hours contracted, with only minor variations in the time allotted for work.

However, in this year's data, we see more diversity in the hours contracted, as shown in Figure 2.

Figure 2: Hours Contracted by Assessment Type

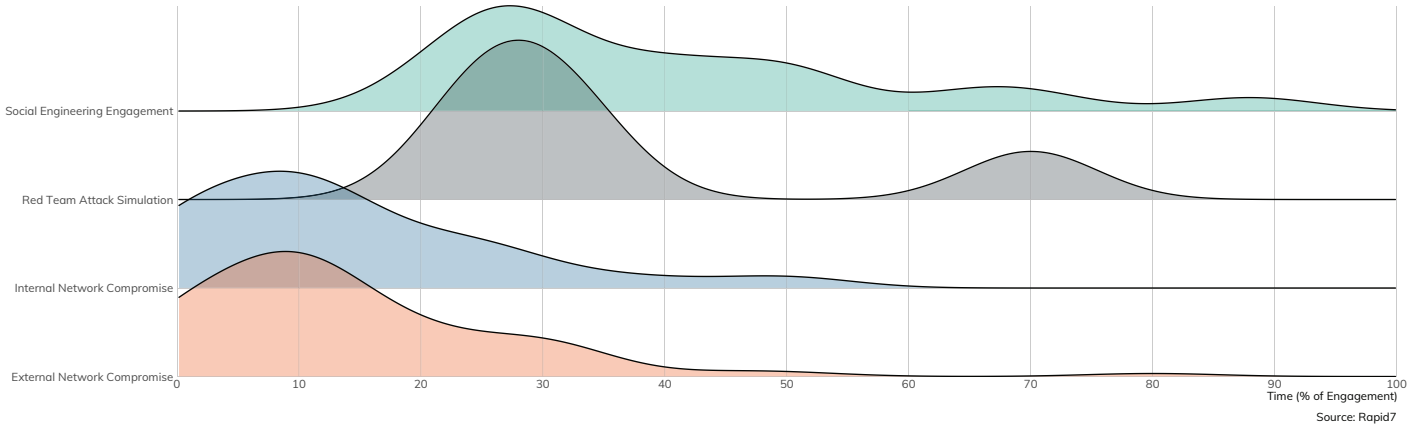


Here, we can see that while there are clearly preferred chunks of time, there are quite a few categories of total times across all engagement types. It turns out, each client engagement is pretty unique, and there really isn't a one-size-fits-all time box that we can conveniently stuff every engagement into. In fact, this diversity in hours contracted speaks to the importance of those initial scoping and planning calls between the client and the penetration tester—any given engagement may be about 35 hours worth of work, all the way up to 150 hours. It all depends on the complexity and size of the network, as well as how deep the client wants to go.

What doesn't change too much is the percentage of time spent planning and performing reconnaissance on a given client's network. In the more complex engagement types of social engineering and red teaming, as much as 25% of the total time is devoted to these pre-attack activities, while internal and external network assessments tend to jump right off the blocks with only perhaps 10% of the time spent on recon.

Figure 3: Engagement Time Spent on Planning and Reconnaissance

Displayed set includes only engagements where details on time spent provided (n = 148)



THIS ONE TIME ON A PEN TEST:

I'm Calling My Lawyer!

By Jonathan Stines

As part of a telephone pretexting engagement for a law firm, we were provided with their employee's phone numbers to call in an attempt to identify sensitive information, harvest credentials, and obtain a reverse shell on their machines.

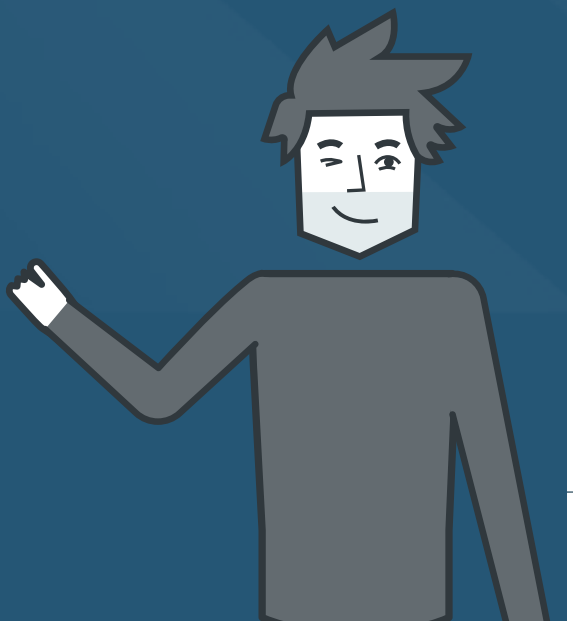
We first started calling personnel posing as an IT analyst, asking general questions about pain points users had been experiencing in order to build trust with the individuals and allow them to voice their complaints. After all, people love complaining about IT! Once this rapport was established, we moved to more pointed questions about VPN technologies in use, endpoint protection and versions in use, as well as other technologies in use that could be used later as part of another pretext.

Now that we knew about their endpoint protection solution's vendor and version, we then crafted a payload designed to evade that specific security tool. Also, with the version of the company's remote access VPN client, we came up with a pretext where we again posed as an IT analyst dialing personnel who were on a list of folks who had outdated VPN clients that needed to be updated.

We called the employees and explained that their version of the VPN client was outdated, had multiple known vulnerabilities, and needed to be updated. We then instructed them to press start, run CMD, and run our stager one-liner. Once this took place, the one-liner would download our hosted payload assembly and run in the user's memory, and establish a reverse-shell to our Command and Control (C2) server, allowing us full access to their workstation.

With internal access to their network, we then performed high-level reconnaissance such as identifying members of security groups such as Domain Administrators and where their Domain Controllers were located. Additionally, we performed a technique known as "Kerberoasting," which returned Kerberos authentication hashes for service accounts on the domain. During testing, we cracked one of the Kerberos hashes for an account that was a member of the highly privileged Domain Admins security group.

At this point, we contacted the customer, explained how far we had gotten, and asked for further instruction. They instructed us to stop making phone calls immediately, as we had sufficiently demonstrated risk from a telephone pretexting perspective.



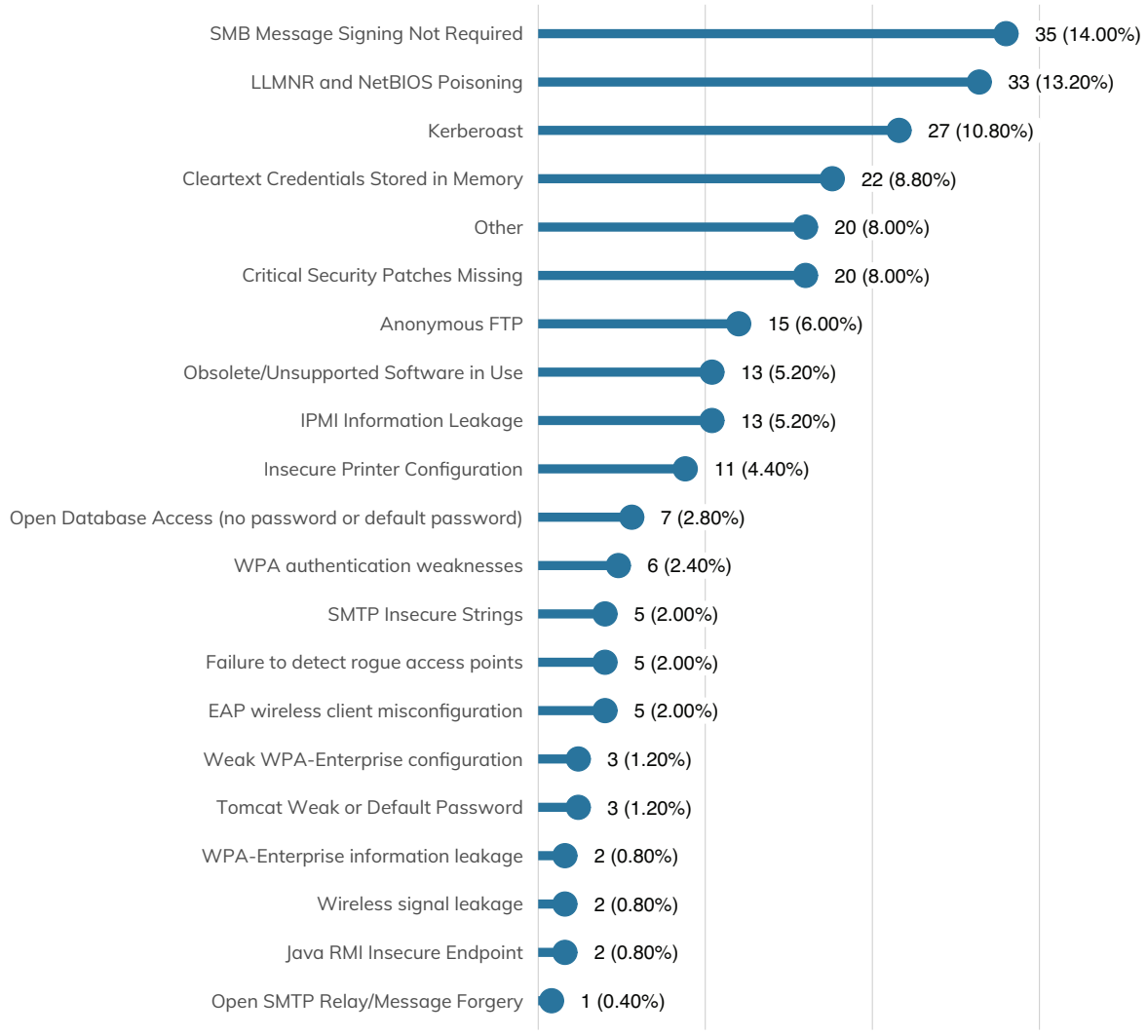
Vulnerabilities Deep Dive

The types of vulnerabilities found and exploited by pen testers depend largely on the type of penetration test being performed. This section compares these vulnerability profiles between inside jobs (internal network assessments and code reviews) and outside jobs (external network assessments, social engineering engagements, and red team simulations).

Inside Jobs

Internal network assessments end up focusing quite heavily on Windows-based attacks, and for good reason. After all, Windows clients abound in the enterprise environment regardless of the industry or size of the target, so aspiring and practiced pen testers alike would do well to make sure their tooling and experience matches with the typical Windows environment. Figure 4 details the most common types of attacks that are successful in an internal environment.

Figure 4: Internal Engagement: What Vulnerabilities Did You Find?



Source: Rapid7

Windows Everywhere

The top three most common vulnerabilities encountered in these internal engagements are **SMB Message Signing Not Required** (14%), **LLMNR and NetBIOS Poisoning** (13.2%), and **Kerberoast** (10.8%). Taken together, this is a powerful set of vulnerabilities that can quickly elevate an attacker to Domain Admin (DA) privileges.

The first, involving SMB message signing, is common in a mixed environment of Microsoft and non-Microsoft SMB clients. SMB message signing is a fundamental cryptographic control that lets SMB clients and servers properly authenticate to each other. However, it's not enabled by default in most situations. Unfortunately for Windows administrators, SMB signing is often not required—or sometimes not supported—by non-Microsoft implementations of SMB. In addition, SMB message signing is only default-enabled on Windows for Domain Controllers, not other Windows clients or servers. While the procedures for enabling SMB message signing are well documented by Microsoft,[2] many Windows Administrators aren't fanatical about enforcing this basic cryptographic control, especially if the asset population of the network is either very dynamic or not well known to the IT operations staff.

Finally, troubleshooting SMB connectivity issues also tends to start with disabling SMB signing temporarily, and temporary troubleshooting measures are notorious sources for persistent enterprise misconfigurations. This is all to say that if SMB signing isn't strictly required by every host on the network, pen testers can take advantage of this lack of cryptographic assurances of destination server identity to get in a position to steal SMB-based credentials, if only they can get these clients to talk to a fake server the client might already have a network relationship with.

That brings us to the second most popular vulnerability, LLMNR and NetBIOS spoofing. If a client doesn't already have cached network address information for a given server on the internal network, they might try to perform a "Link-Local Multicast Name Resolution" (or LLMNR), and whatever machine on the local network responds first is the one that the client will believe is the destination server.[3] Of course, if both the real client and the real server required SMB signing, this impersonation wouldn't last very long, since the spoofed SMB server wouldn't be able to authenticate correctly. However, as discussed above, pen testers often run into situations in which SMB signing isn't required on internal engagements.

So, combining the two issues above gets an attacker in a pretty solid position to start acquiring Windows password hashes. A client, which doesn't require SMB signing, needs to find a server it doesn't have an address for, and that server (for whatever reason), doesn't have a pre-defined DNS entry. It asks the network, "New cache, who dis?" and the attacker's machine helpfully responds, "It me! Who dis?" and challenges the client for a Windows authentication. The client, believing the attacker's server, provides a hash as part of the SMB challenge/response, which can then be cracked later by the attacker and reused to log in to the client directly.

Once the attacker has access to a Windows domain user's account, they typically can log in to any workstation in that domain—specifically, workstations that may be running services that need a Kerberos-enabled service account to run. Enter the "Kerberoasting" attack, where an attacker on a local machine can peek at the running processes and perhaps get access to an encrypted (but crackable) version of a domain service account's password.[4] Service accounts, like those that run domain backups or antivirus, tend to run with pretty high privileges, and if their passwords are set by humans, they tend to be pretty crackable in short order.

Finally, some services don't take advantage of domain-based Kerberos tickets for authentication, as indicated by the No. 4 vulnerability encountered, **Cleartext Credentials Stored in Memory.** Again, if an attacker is able to log in to a domain-connected workstation, those credentials are merely a Mimikatz[5] session away.

---

[2] https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/microsoft-network-client-digitally-sign-communications-always

[3] NetBIOS works in basically the same way (using local network broadcast requests), to the point of being functionally identical, which is why both techniques tend to be lumped together.

[4] How this all works is actually pretty complicated, but well explained by Rob "Mubix" Fuller in his three-part blog series, starting at: https://room362.com/post/2016/kerberoast-pt1/.

[5] https://github.com/gentilkiwi/mimikatz, first released by Benjamin Delpy, and reimplemented in nearly every penetration testing framework.

Not Just Windows Misconfigurations

Of course, most internal networks have several types of operating systems and networked software running, not just the standard, default configurations of Microsoft clients and servers. For these situations, pen testers can seek out software with **Critical Security Patches Missing** (leveraged 8% of the time in this study) or running **Obsolete or Unsupported Software** (present in 5.2% of engagements), and draw on the thousands of known, pre-existing proofs-of-concept or working exploits available to them.

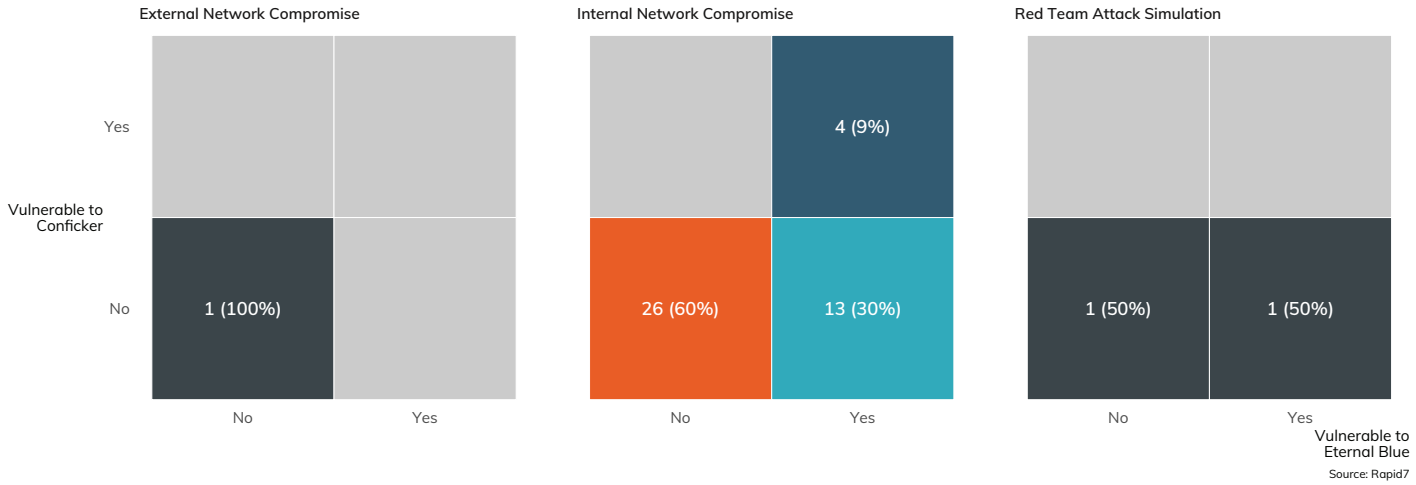
MS08-067 and MS17-10

Two vulnerabilities stand out as pretty standard go-tos for any internally scoped network assessment: MS08-067, which was weaponized in the Conficker exploit back in 2008, and MS17-10, which was the central vulnerability to the EternalBlue exploit kit of 2017. These two issues are among the famous vulnerabilities of the past decade, so you would think that IT and IT security teams would have long ago excised these vulnerabilities from their internal networks.

Unfortunately, that's not the case, as Figure 5 shows:

Figure 5: Were Any Hosts Vulnerable to EternalBlue and/or Conficker?

Count reflects only provided responses. Percentages calculated within assessment types.

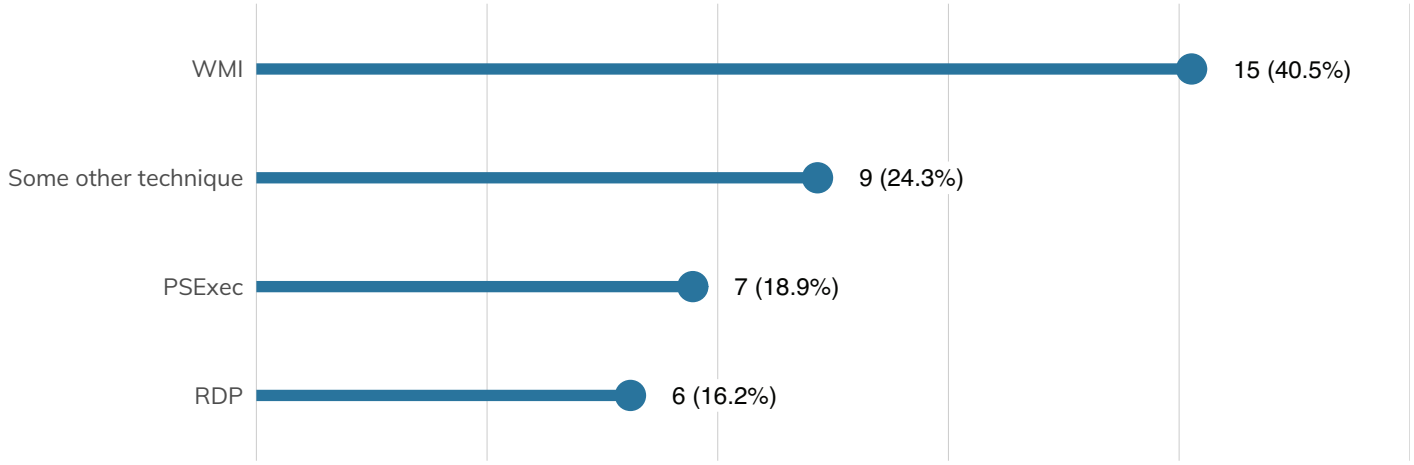


The Unknown Unknowns

Patch and asset management is a critical function of IT operations, and exploiting those unpatched and antique systems is often seen as the first and primary job of the penetration tester so that IT ops can know where their patch management program has blind spots. While this is certainly an important aspect of penetration testing, the fifth most popular vulnerability is the ever-elusive **Other** (even with missing patches at 8% of the time). This category, and its fifth-place showing, demonstrates the true value of penetration testing. So far, we have not figured out how to totally automate security controls (or testing those controls), and so we rely on the creativity and situational awareness of the savvy pen tester to illustrate new, novel, and unanticipated vulnerabilities in the internal network.

Figure 6: Internal Engagements: What Lateral Movement Technique Did You Leverage?

Set only includes cases where lateral movement was achieved.



Lateral Movement Techniques

Once a pen tester has achieved access to at least one machine in the network, the next job is typically to leverage that access to move around the network. There are a few ways to perform this sort of lateral movement, as we can see in Figure 6.

Two of these techniques, WMI and PSExec, use built-in Windows system interfaces to extend control around the network. They are commonly used in both normal network maintenance and as techniques leveraged by the notorious WannaCry and NotPetya self-propagating worms of 2017 and 2018. Most discussions of these worms focus on the EternalBlue exploits implemented[6] but don't talk about the real reason why these worms are so effective: They use the same sophisticated techniques for lateral movement as real attackers and pen testers alike. By recovering and reusing passwords on compromised systems, attackers can often flit from machine to machine in search of their ultimate targets.

Code Review Corner

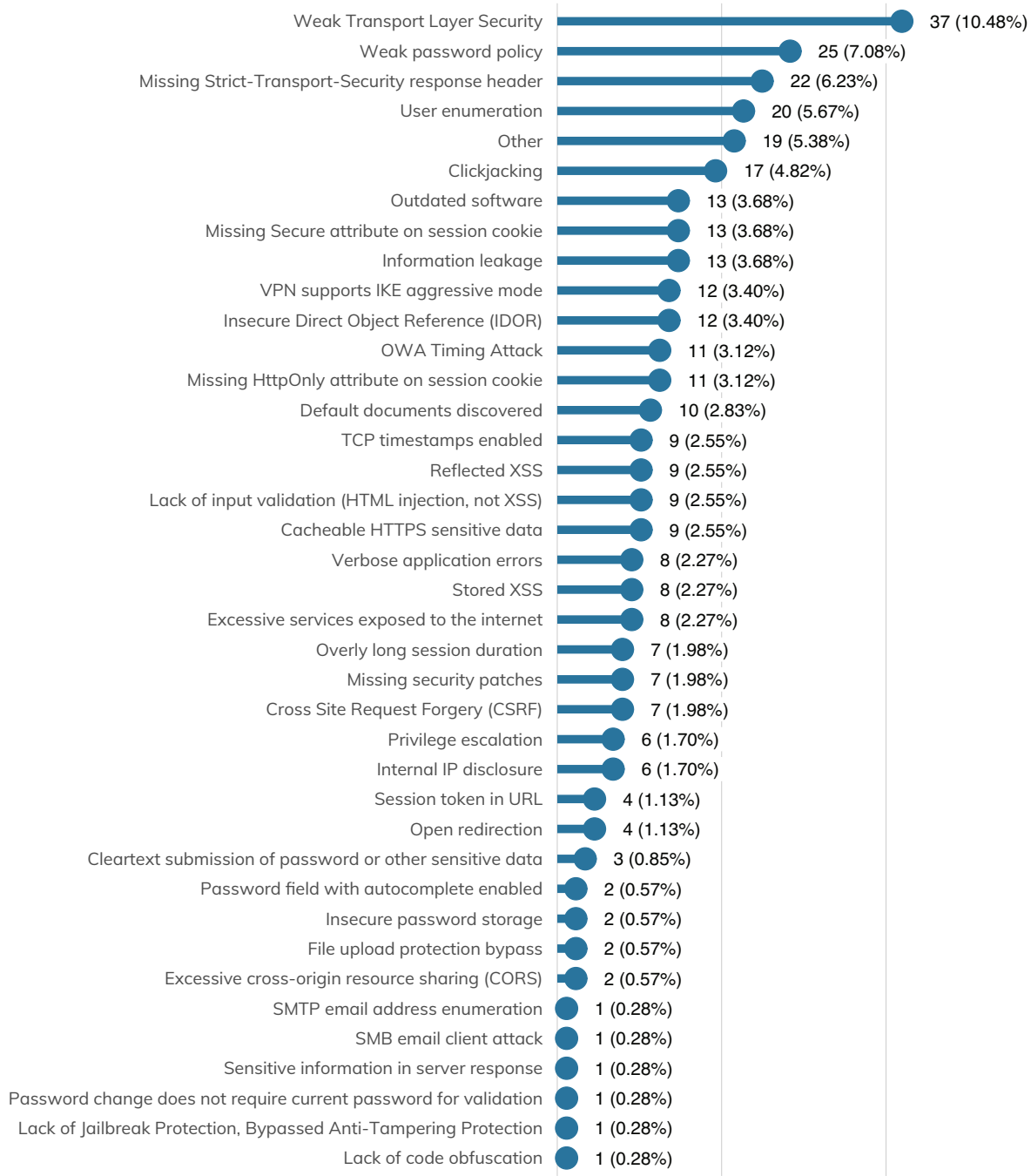
Since code review assessments only comprise about 3% of our total corpus, we don't really have much statistically relevant information about the types of vulnerabilities that are uncovered in code review. That said, the vulnerabilities we did find covered the spectrum of typical web application vulnerabilities: SQL injection, reflected XSS, insecure data transfer and storage, CSRF, and all the rest. In other words, they looked pretty much just like external web application-centric pen tests (see the next section for more details on those). With so few examples, though, it's difficult to say whether this is normal or not.

[6] Including in this very paper, just a few paragraphs ago!

Outside Jobs

Most penetration tests at least start off externally based—about 65% or so, as detailed in Part 1 of this paper. Thus, these explorations focus heavily on web-based technologies that are more or less platform-independent, as we can see in Figure 7:

Figure 7: External Engagement: What Vulnerabilities Did You Find?



Source: Rapid7

These vulnerabilities also aren't what we would normally consider "high-severity" vulnerabilities (those ranking at 8.5 or above on the Common Vulnerability Scoring System). Such vulnerabilities nearly always have some kind of remote code execution component, but the top vulnerabilities encountered by external penetration testers are:

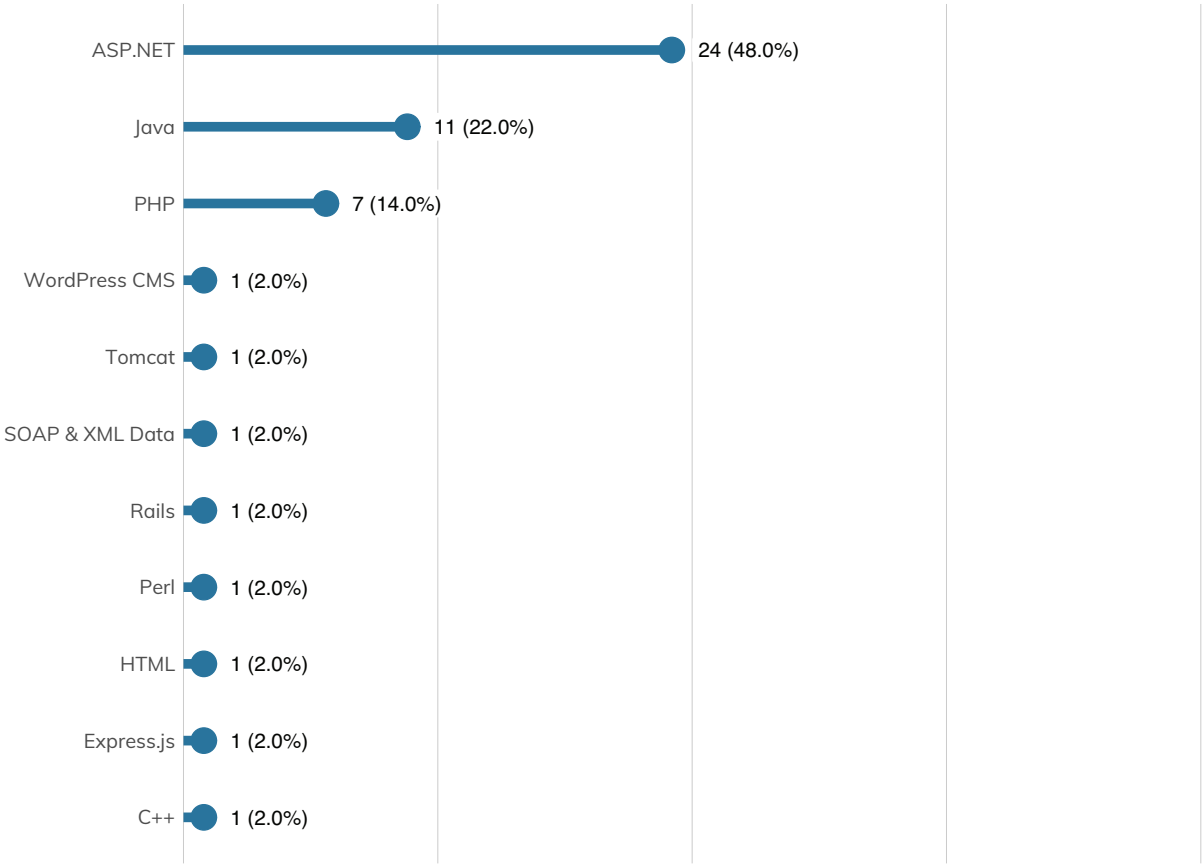
- Weak Transport Layer Security (10.48%)
- Weak Password Policy (7.08%)
- Missing Strict-Transport-Security (STS) response headers (6.23%)
- User enumeration (5.67%)

All of these vulnerabilities are more information leak-oriented, and require some other action on the part of the attacker—either intercepting normal communication due to weaker cryptographic controls, or a credential-stuffing or password-spraying based attack to at least gain post-authentication privileges to an application. It isn't until we get down to **Other** (5.38%), **Outdated Software** (3.68%), and **Insecure Direct Object Reference** (IDOR) (3.4%) where we start seeing more severe, straight-shot-to-command-execution vulnerabilities and exploits.

Web Application Technologies

Once an external pen tester discovers an issue with a web application, it's often useful to have a passable, working knowledge of what that web application was written in.

Figure 8: What Technology Was the Web Application Built On?



Source: Rapid7

For external and internal applications alike, we see that **ASP.NET** (48%) tops the list, followed by **Java** (22%) and **PHP** (14%). This may come as sad news to the web app developer who is considering a career in web app penetration testing, since these web app technologies don't fare very well on StackOverflow's recent "Most Loved Languages" survey.[7]

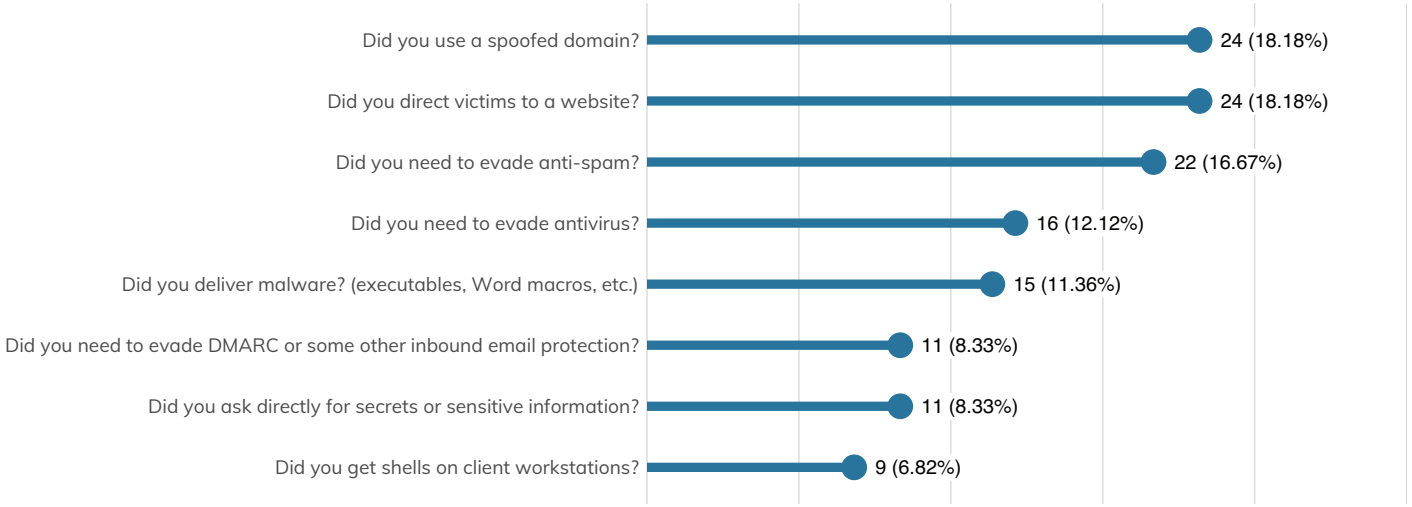
Indeed, the hottest web app frameworks like Rust, Kotlin, and TypeScript rarely come up in enterprise penetration tests (yet). This tells us that enterprise IT, at least when it comes to web application development, has some catching up to do with the avant garde, and could certainly benefit from an infusion of these technologies that happen to not just be more loved, but are fundamentally more modern and type-safe, having built-in backstops to prevent many common web app vulnerabilities.

Electronic Social Engineering

Many advanced external engagements involve electronic social engineering (ESE). While these tend to exploit "Layer 8" vulnerabilities (those involving aspects of human interactions and tendencies like trust, fear, or greed), there is still a significant technological component to such attacks, as Figure 9 illustrates:

Figure 9: Email-Based Electronic Social Engineering: What Happened?

Percentages calculated based on aggregated total count of attempts.



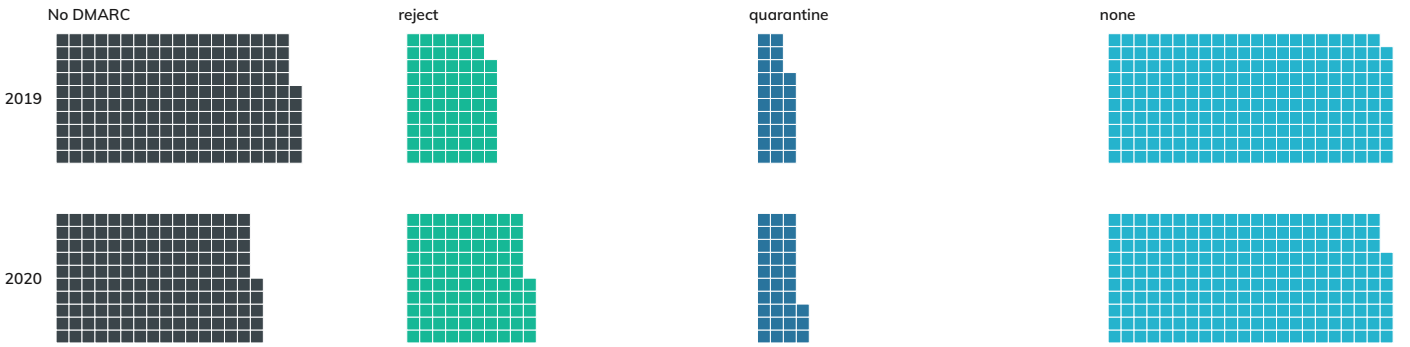
Source: Rapid7

Spoofed domains show up in 18% of ESE survey responses, and only rarely did respondents need to **evade DMARC** (8.33%). This illustrates how important anti-spoofing defenses are in protecting employees from falling for these attacks. DMARC, or Domain Message Authentication Reporting, makes it difficult for phishing emails to purport to come from within the targeted domain, which means attackers need to craft more convincing phishing lures using other, easier-to-spot techniques. Indeed, Rapid7's recent re-assessment of the Fortune 500 and how their DMARC status fares[8] shows some positive movement.

[7] https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-languages-loved (Note that while ASP.NET is not specifically surveyed by StackOverflow, most ASP.NET applications are written in C#.)

[8] https://blog.rapid7.com/2019/10/18/what-a-difference-a-year-makes-revisiting-our-inaugural-fortune-500-icer-one-year-later/

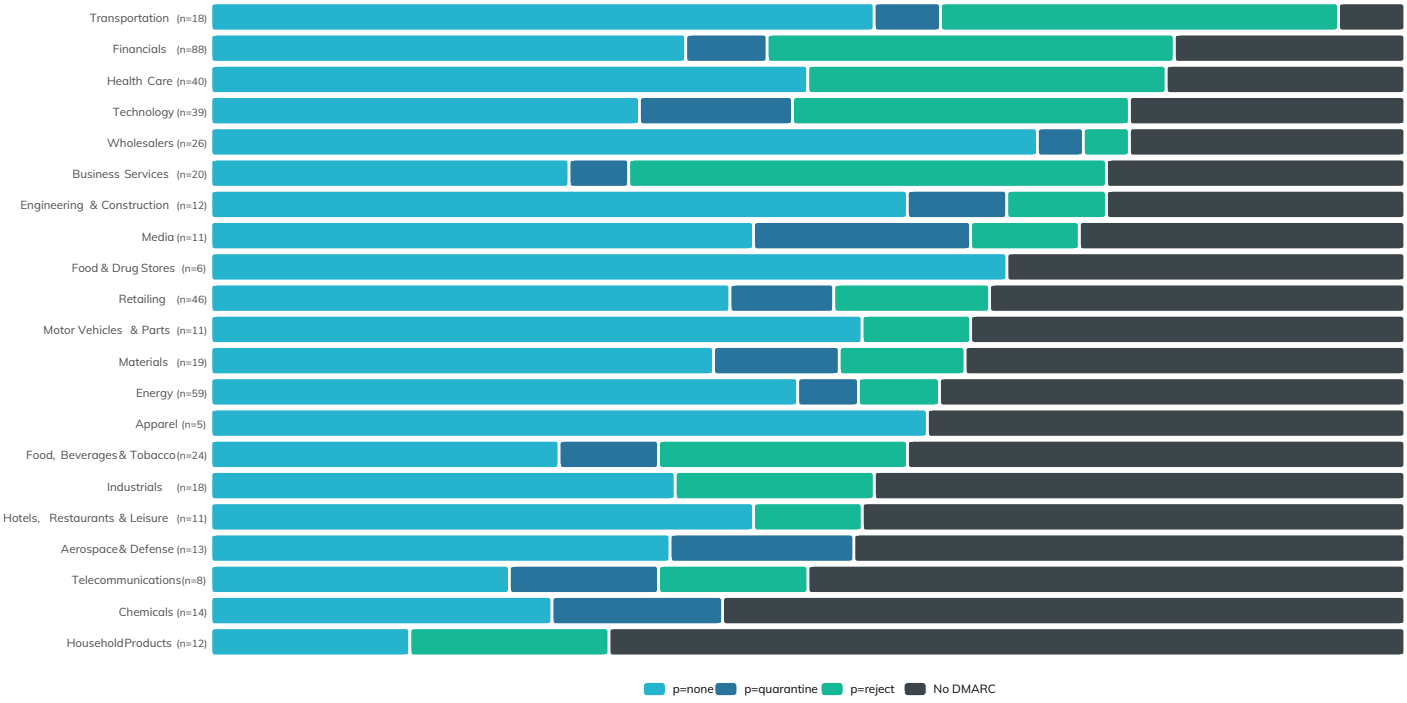
Figure 10: Fortune 500 DMARC Status: 2019 vs. 2020



Source: Rapid7

From 2019 to 2020, we've seen a 20% increase in Fortune 500 sites that went from "No DMARC" to at least a DMARC policy set to at least one of "none," "quarantine," or "reject," and a 40% increase of "reject" DMARC policies over the same period. While it's great to see that DMARC deployment is chugging along, it does appear that these benefits aren't being distributed evenly over all industries, and some industries are still just getting started with "none" policies.

Figure 11: Fortune 500 DMARC Status



Updated: August 2020

Also, the most common "payload" for a ESE attack is to direct users to **an attacker-controlled website** (18.18% of the time), which is much more common than **delivering malware** (11.36%). We expect that using websites as intermediaries to collect passwords and attack browser sessions will be more successful over time, due in part to browser manufacturers pushing for browser-controlled, encrypted DNS (which escapes inspection by an enterprise IT security authority),[9] and the pandemic-created diaspora of knowledge workers around the world (who are much more likely to read email and browse the web outside of an enterprise-controlled VPN).

[9] https://spectrum.ieee.org/tech-talk/telecom/security/the-fight-over-encrypted-dns-boils-over

Red Team Simulations

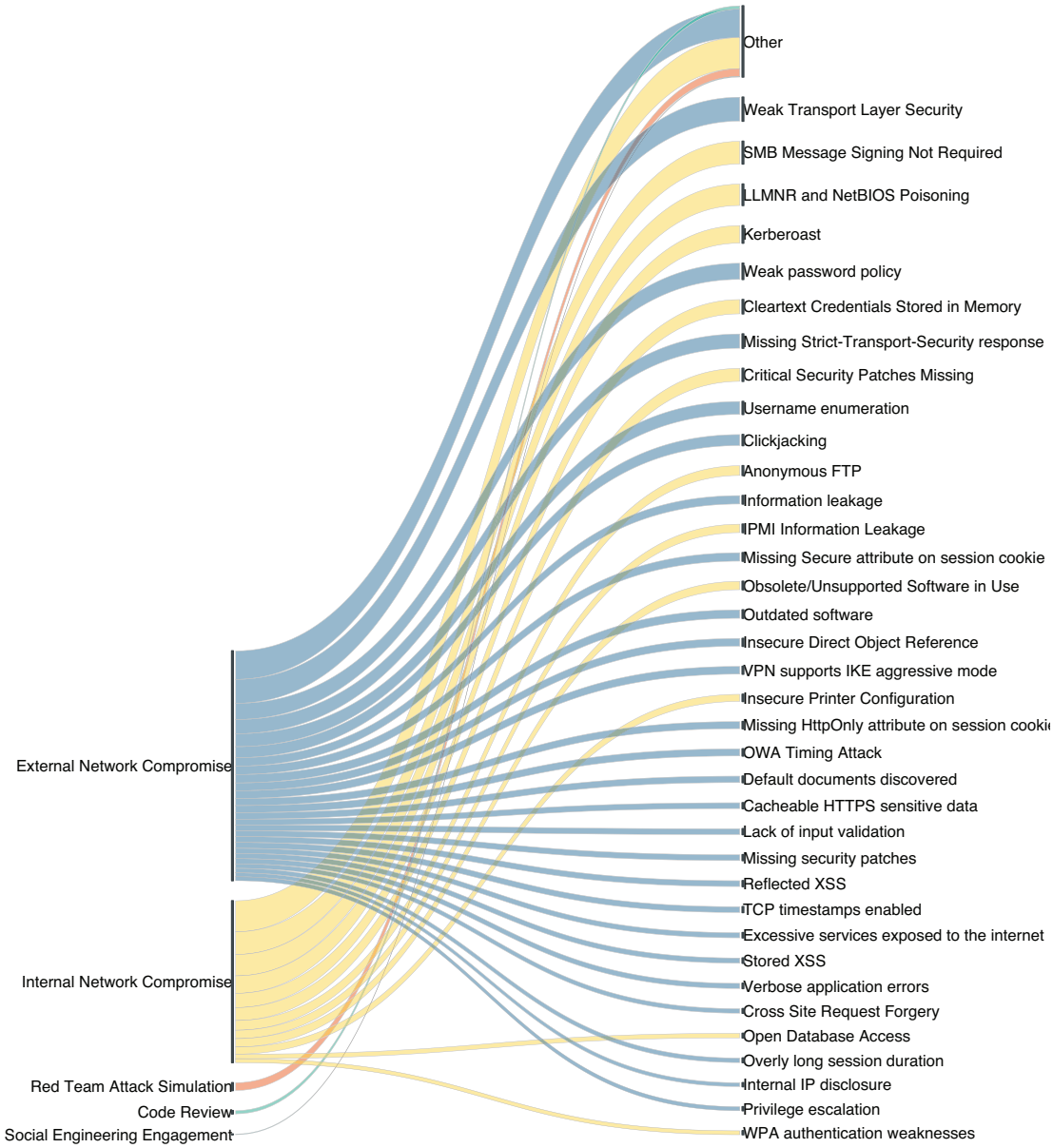
Like the Code Review category of inside jobs, red team simulations are fairly low-count in our collected data, and so, it's difficult to uncover any statistically significant findings from their engagement reports. That said, the No. 1 vulnerability type uncovered in these simulations is "Other," which speaks to the fact that these engagements tend to require a fair amount of situational awareness and on-the-spot creativity to complete effectively. We suspect this is because the kind of customer that signs up for a red team engagement is already pretty far along the spectrum of security maturity, and has their business locked down pretty tight.

Vuln Popularity: What to Defend

Regardless of assessment type—inside job or outside job—it can be illuminating to see which vulnerabilities rise to the top of the overall list of those reported by our penetration testers.

Figure 12: Vulnerabilities by Engagement Type

"Other" includes vulnerability types that had fewer than five occurrences. Vertical bands correspond to occurrence counts.



Here, we can see that **Other** is the most common vulnerability type, which further underlines the assertion that penetration testers can't just go through the motions of network compromise. Each network is unique in its own way, and a successful penetration test relies on the expertise of a seasoned and dedicated penetration tester noticing when something seems off.

Aside from those unknown unknowns, IT operations teams would do well to survey their own networks for the most common vulnerabilities exploited before the penetration tester arrives on the scene. This is especially true in industries that have a regulatory requirement for regular penetration tests. Preparing for a predictable engagement by taking care of these basics not only gives the penetration testers more time to focus on the less-explored areas of your network, but also hardens the enterprise environment against the most common types of attacks that criminals exercise.

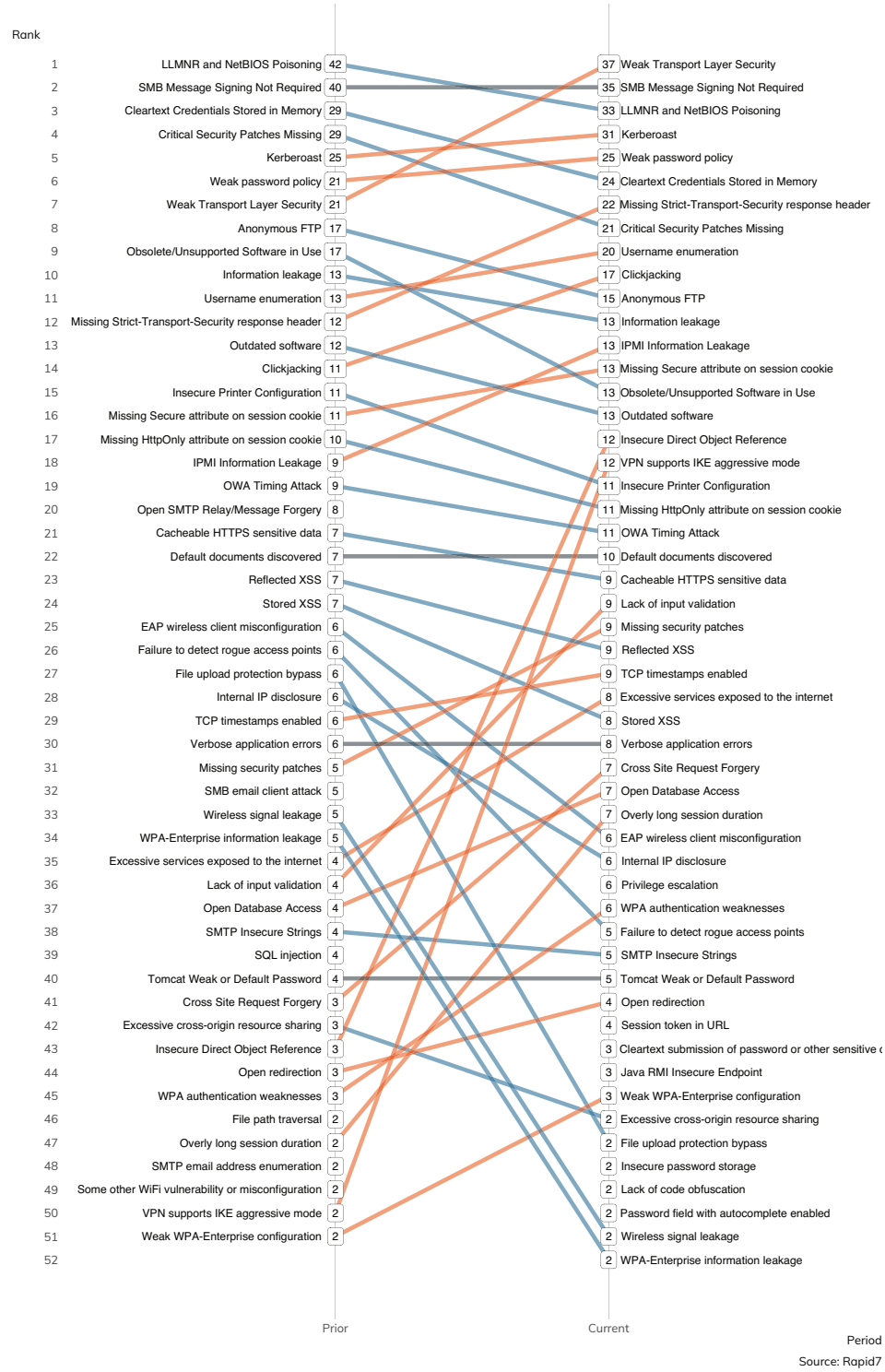
Year-Over-Year Changes

This is the second Under the Hoodie report Rapid7 has produced using the same vulnerability questionnaire, which gives us an opportunity to see how individual vulnerabilities are faring, year-over-year, against our penetration testers.

The two steepest increases involve **VPN supports IKE aggressive mode** and **Insecure Direct Object Reference (IDOR)**, which is telling. The VPN-based investigations almost certainly reflect the new, mid-pandemic reality of VPN reliance to get a suddenly huge population of stay-at-home workers online and productive, while the IDOR exposures similarly reflect an increase in attention on how web applications are written and how they can be exploited to run arbitrary code from an external position.

Figure 13: Prior vs. Current Vulnerabilties

Ordering based on in-period relative frequency. Numeric labels reflect count of vulnerability instances.



Source: Rapid7

THIS ONE TIME ON A PEN TEST:

Doing Well With XML

By: Tommy Dew

Most recently, I was on what appeared to be a normal web application penetration test. Typically, our clients will provide us with a few user accounts, such as an admin account and maybe a few different user roles. They'll also give us any relevant API documentation and examples, or design information. However, one thing that stood out to me in the documentation of the application was how by design, the application specifically allowed for uploading XML files. Once I began going through the usual web application testing methodology and got a feel for the application, I began creating specifically crafted XML files that would allow for XML External Entity processing.

If you're not familiar with how XML External Entity processing attacks work, it goes something like this: A penetration tester, or malicious actor, will target application functionality that allows for the processing of malicious XML data by a weakly configured XML parser. If the malicious XML is successfully processed by the XML parser, the tester may introduce external entities into their payload, which could allow them to retrieve the web application server's system files (think /etc/passwd), or perform Server Side Request Forgery (SSRF) attacks, instructing the web application server to make any HTTP request to any desired URL, given there isn't any network egress filter server-side.

Since I knew that the web application accepted XML, I wanted to see whether it would blindly accept malicious XML that would make a request to a Rapid7-controlled web server. First, I crafted an XML file that would simply instruct the web application server to make a GET request to me. On my jump box, I fired up a simple temporary web server, then submitted the malicious XML to the web application. Instantly, I received an 'GET' request from an IP address. After a quick verification, I confirmed that it was the web application server's IP address!

After knowing that the web application accepted my malicious XML file, I wanted to demonstrate additional risk and see whether I could perform additional actions, such as the SSRF attack mentioned previously. After a quick discussion and recommendations with some team members, I again crafted a new XML file, but this time, I defined the XML file to instruct the web application server to attempt the retrieval of a nonexistent file on my jump box. The web application server was Windows, so we knew that if the attack was successful, it would also transmit Windows NTLM hashes to me. Next, I fired up Responder, a program that allows for capturing Windows hashes, on my server and uploaded the payload to the application, which then successfully transmitted the web server's hashed Windows credentials to me!

However, the credentials were for a Windows machine account, which are notoriously difficult to crack during a reasonable time of a web application penetration test. However, my client responded positively and worked with us to resolve the vulnerability in a couple of days.

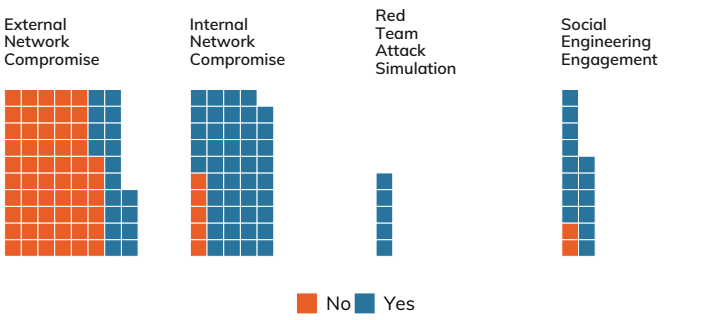
While we do not come across this type of vulnerability in every application assessment, you can bet that it's something we always test for, especially if an application allows for uploading content or submission of XML through an endpoint. We believe that the demonstration of risk by combining XML with SSRF also translated the level of severity to the client and allowed them to dedicate resources to the application to resolve the issue immediately. This is one of the most rewarding aspects of penetration testing, when you can see your work making a positive impact and improving security postures, where possible.

Collecting Creds

While exploiting vulnerabilities and misconfigurations is at the heart of any penetration test, just as critical is how well the credentials used in an enterprise are protected. After all, while popping shells with a carefully crafted buffer overflow is great fun, it's often easier to simply impersonate an authorized user to perform otherwise unauthorized actions. In addition, the purpose of many straight technical attacks is to recover a key password and assume a role of Domain Administrator. Figure 14 shows just how often credentials are compromised in a given scenario.

Figure 14: Did You Capture Any Credentials?

Subsetted to cases where credential capture was in scope for the engagement.



Source: Rapid7

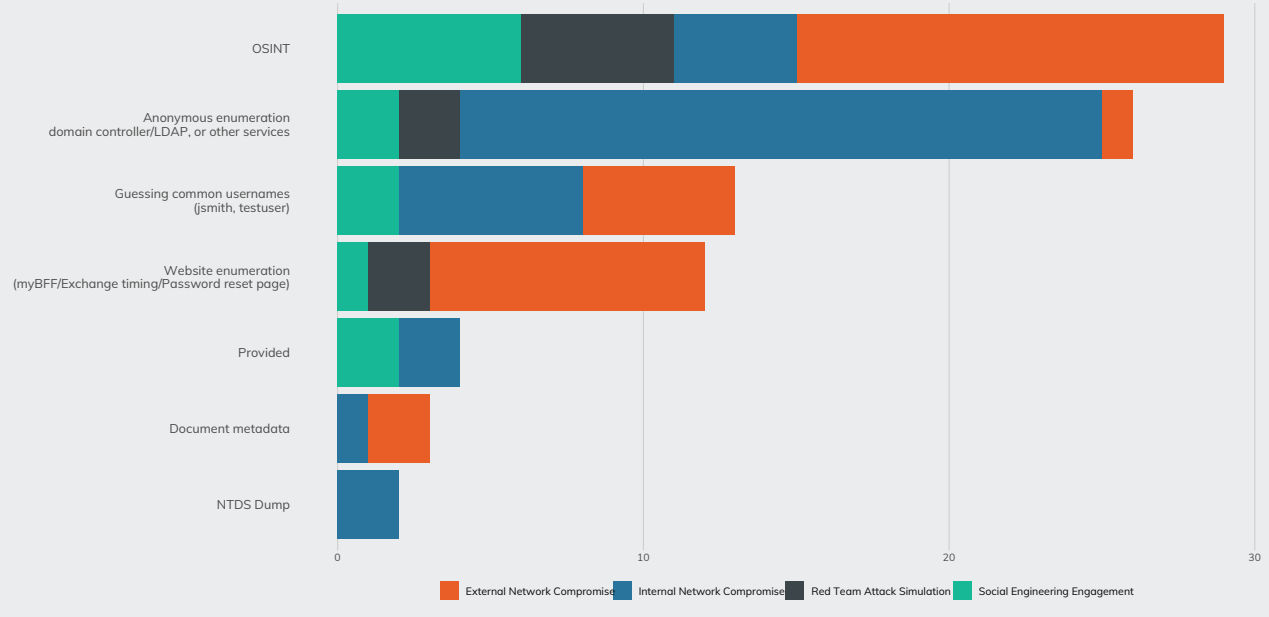
Here, we can see the stark difference between inside jobs and outside jobs; external penetration tests often aren't concerned with credential theft, while internal network compromises are normally all about the credentials. We can see, too, that the more advanced engagements involving red team attack simulations and social engineering engagements are nearly always successful in purloining passwords.

Understanding Usernames

Typically, the first step in collecting credentials is acquiring a list of valid usernames. In Figure 15, we can see the various methods attackers use to recover this half of a working credential.

Figure 15: Credential Capture: How Did You Gather Usernames?

Techniques reflected occurred in aggregate more than one time.



Source: Rapid7

Now, usernames, nominally, aren't secret, and are sometimes technically "public." After all, if nobody knows your username, they can't email you, assign you document permissions, or anything like that. That said, we can see that **OSINT** (or, open source intelligence) is often used to gather the likely usernames used in a given enterprise—that would include social networking sites like LinkedIn and Facebook, where employees can share, wittingly or not, their names, usernames, and places of employment. On internal engagements, we see that attackers usually use internally public, **anonymous enumeration** sources, such as a domain controller or LDAP service.

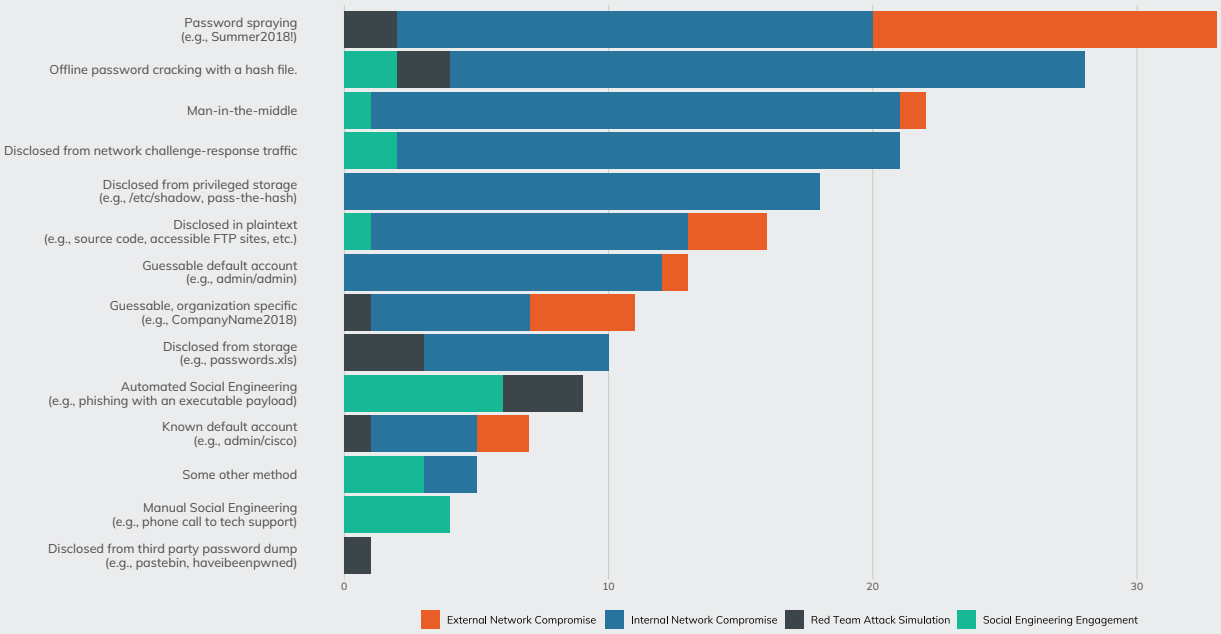
We don't think we're at a point yet where we can say definitively that usernames should be treated as internal secrets, but we can say that enterprises shouldn't make it easy for attackers to build a list of valid usernames. Indeed, the second most popular way to collect usernames from the outside is through **website enumeration**, which is when a web application gives away information on what is a valid username and what isn't. While controlling public OSINT is difficult for any but the most secretive organizations, discovering and disabling anonymous enumeration sources through overly permissive domain controllers and web applications can make an attacker's job much more difficult.

Purloining Passwords

Of course, passwords are supposed to be secret; they're the one secret everyone in an organization is supposed to keep just for themselves so they can access the shared, internal secrets that make an organization function. However, there are myriad methods to acquire these passwords, especially when they're generated by humans and their woefully unoriginal meat brains.

By far, the most common way to "guess" a password, either on an internal or external engagement, is to make sure your guesses are especially accurate. The No. 1 method is **password spraying**, in which the attacker already knows (or has a good idea) of a list of valid usernames, but tries only a few of very special, unoriginal passwords. This technique is much like the old huckster's technique of "cold reading," where the "psychic" appears to make some very intimate assertions about the subject. In fact, though, they're relying on knowledge of the general human condition—nearly everyone has a lost love with the first initial "M," a dead relative with a first initial of "J," and a desire to address some unfinished business with either.

Figure 16: Credential Capture: How Did You Obtain Passwords or Password Hashes?

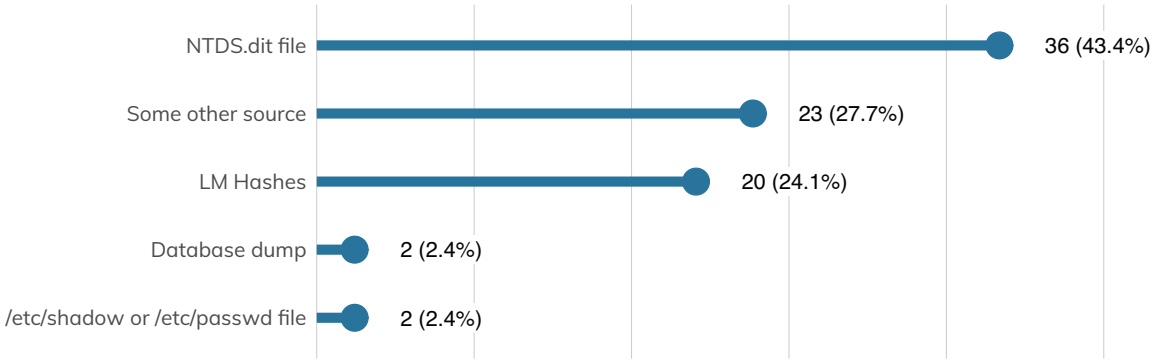


So too, people are constantly inventing and reinventing password schemes that comport with their company policy: It must be at least eight characters, have some numbers and a bit of punctuation, be easy to type, be easy to remember, and be easy to change on a 90-day schedule. And nothing fits that bill better than "Summer2020!"[10]

Hacking Hashes

The second most common method of obtaining good passwords is by first collecting a set of hashes, or encrypted passwords, then throwing huge dictionaries along with processor power at them. In fact, the sources of hashes are somewhat varied, as we can see in Figure 17:

Figure 17: What Kind of Hashes Did You Collect?

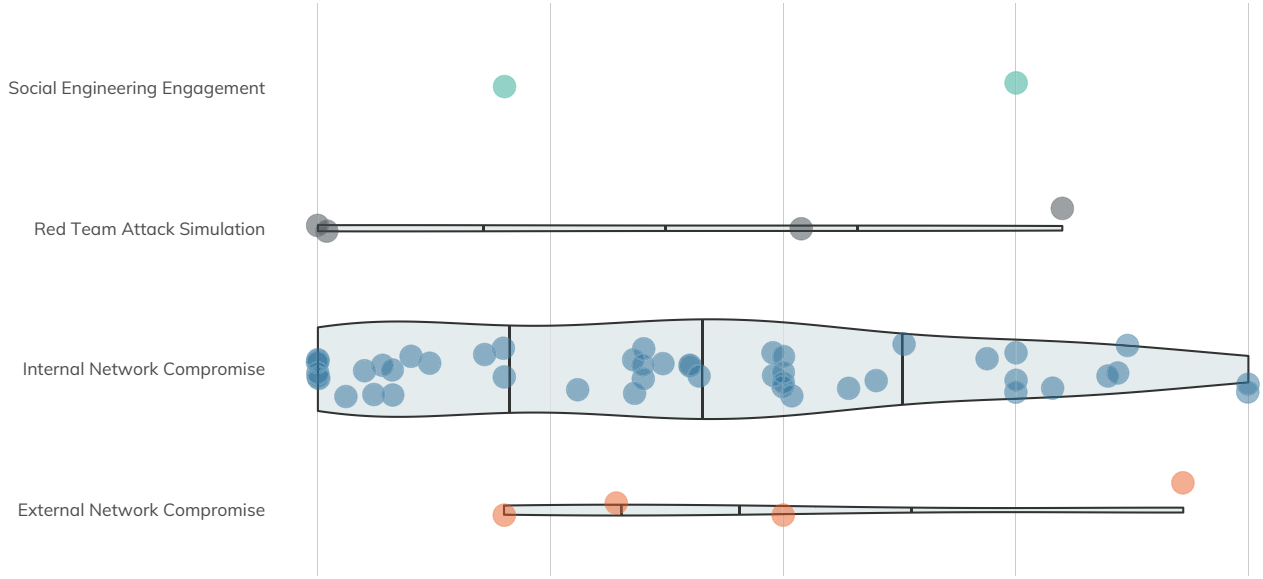


Source: Rapid7

Collecting password hashes is commonly a first-order target of an internal network assessment, since they're so useful, but passwords are hashes precisely so they can't be simply read and used by attackers. However, should the hashfile fall into the wrong hands, it's the unfortunate truth that they tend to be full of easily cracked passwords. Figure 18 shows that when a hashfile is obtained, some proportion of passwords are cracked within the agreed-to engagement time.

Figure 18: About What Percentage of the Passwords Did You Crack?

Outlines correspond to distribution. Line breaks correspond to the 25%, 50% (median), and 75% quantiles.

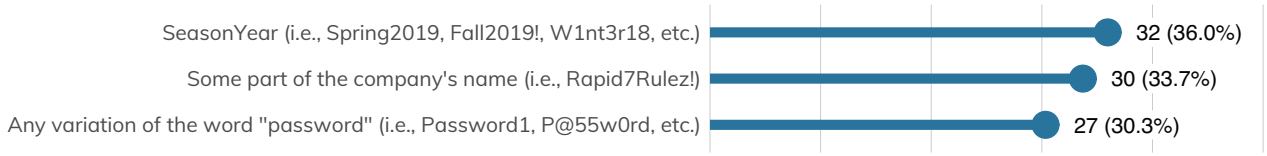


[10] If this is your password, change it now. Really. And not to "Autumn2020!"

Not only do we see that truly strong passwords are rare in a given hashfile, but we can also confirm that people pick very guessable, predictable passwords when given half a chance:

Figure 19: Guessable Cracked Passwords

Set limited to engagements where password cracking was in scope.



Source: Rapid7

Watch Out for Null Sessions

One popular method for confirming guessed or cracked passwords is by leveraging Windows Null Sessions. Once upon a time, null sessions were quite common in Windows-based LANs, but in recent years, Windows administrators have realized that they don't offer much other than this handy service for attackers. Today, only 40% of surveyed penetration testers found null sessions enabled on domain controllers, which is down from the half-and-half reporting from 2019.

Today, anonymous null sessions are either disabled or their state is undefined on new Windows machines.[11] Because they're not disabled by default across the board, Windows domain administrators need to go out of their way to disable them.

Figure 20: Did Any Domain Controllers Have Null Sessions Enabled?

Source: Rapid7

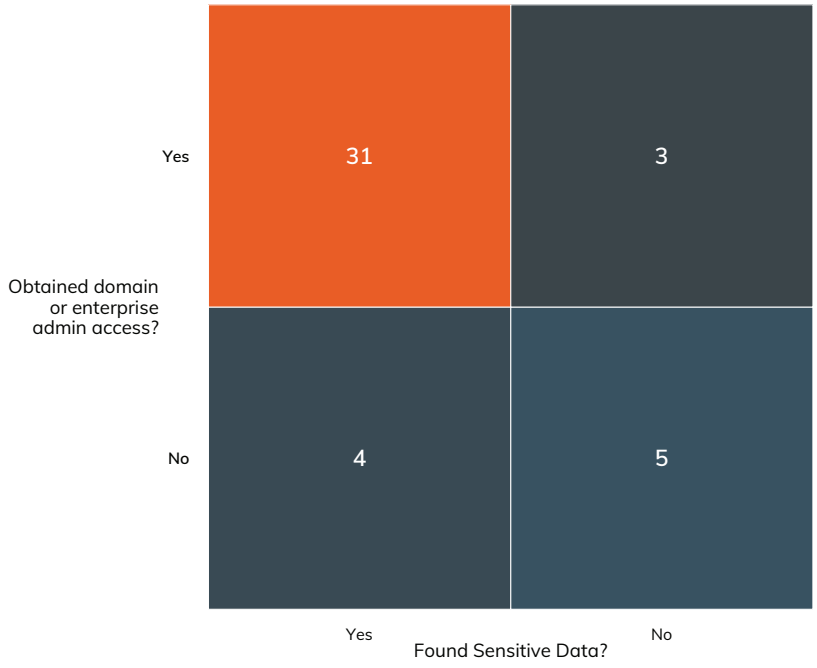
[11] https://docs.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/network-access-restrict-anonymous-access-to-named-pipes-and-shares

Privileged Accounts and Sensitive Data

While regular user accounts are often the way into a given organization, the most sensitive data is often beyond the reach of that one user with a guessable password. The easiest way to obtain sensitive information is to escalate to the level of a Domain or Enterprise Administrator, and in 31 internal network compromises, the one leads to the other, as shown in Figure 21.

Figure 21: Internal Network Compromise: The Path of Least Resistance to Sensitive Data

Privileged accounts provide the keys to unlock the credential vaults. Value in cell represents enagement count. Only cases with responses presented.



Source: Rapid7

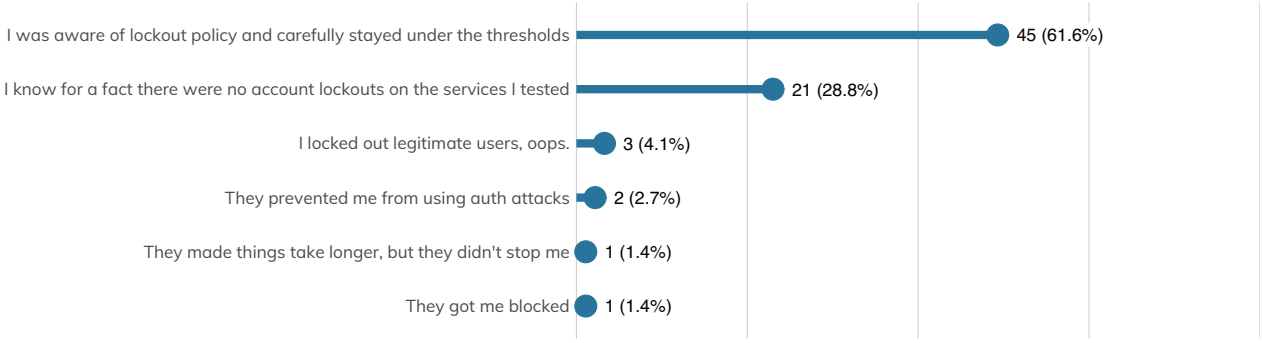
Lockout Policies and 2FA

Lockout policies and two-factor authentication (2FA) are important security controls to both protect passwords and limit their utility once discovered by attackers. Unfortunately, their deployment is scattered or otherwise ineffective in today's enterprise environments. Figure 22 is especially troubling.

Here, we can see that only a small minority of external engagements were hindered significantly by account lockout policies. This is not to say lockout policies aren't effective on the services they cover; rather, the lockout policies either are ineffective because the penetration tester is spraying a small number of passwords across all users (thus, staying below the threshold for each tested user), or there were services available—usually email—that did not enforce the lockout policy employed by the main authentication system.

Figure 22: External Engagement: How Effective Were Lockouts?

Count reflects only available responses.



Source: Rapid7

That said, our surveyed pen testers did run into 2FA more often this year than last year:

Figure 23: Was Two-Factor Authentication Enabled on Any Services You Tested?

Source: Rapid7

In our 2019 survey, we saw 2FA only 21% of the time, while this year, we see a significant growth to 35%. Again, in order to be effective, 2FA needs to cover every egress point, which means that all secondary authentication systems either need 2FA, or they need to employ a different, unique password.

Securing Credentials

The best defense against both password spraying and protecting the contents of hashfiles from commodity cracking is, somewhat surprisingly, the same: Don't rely on humans to pick passwords. The one program any IT organization can take on to best enhance the security of their organization would be to standardize on **machine-controlled password management.** There are many solutions to choose from, but ultimately, you want some kind of mechanism to generate, and save, strong, long, unique passwords, and get *everyone* in the organization on board with it, both at work and at home. Nearly all offensive operations, either simulated through a penetration test or conducted for real by actual criminals, involve guessing, cracking, or otherwise obtaining a critical password at some stage of the process.

THIS ONE TIME ON A PEN TEST:

Outwitting the Vexing VPN

By: Robert Stewart

Just as we do with the vast majority of our engagements, I started by digging around the internet looking for information about my target's employees. One thing I had a hard time figuring out was what their username format was. I scraped the metadata from documents they hosted, used resources like hunter.io, and also used the Harvester to dig up what I could. But, I was still not positive on the format. Most companies use common formats like FLast, or First.Last. To validate the username formats, I was using a tool called lyncsmash, which leverages a timing vulnerability in the Lync service used by on-premises Skype servers. Testing FLast, and First.Last turned up nothing for me, but on my third try, I thought to use LastF, and that turned out to be the ticket. I start seeing a fair amount of valid usernames scroll by.

I parsed my list of usernames for valid accounts and found that out of 482, I ended up with 230 valid accounts. This was still a good number, so I kept rolling with a password spray using lyncsmash. After about my fourth try, I hit some gold: Eleven accounts came back, all using the same weak and guessable password.

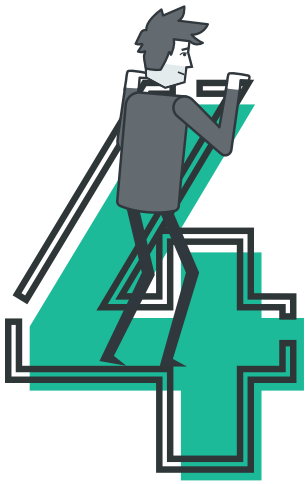
You can't hate on having 11 accounts to validate external access with, am I right?!

While doing my OSINT, I found several external services that my target used for things like email, VPN, and some other remote access stuff. But then came the low: They were using multi-factor authentication (MFA) on every external portal. Ugh. So, I started thinking about it and realized that their VPN has several different authentication profiles. So, I tested all of them, just to be sure I didn't leave a stone unturned. Unfortunately, no luck.

I went back to doing more OSINT, hoping to find something I could log in to with these accounts. And out of the darkness, I found it: a second VPN endpoint. And, this one had an extra authentication profile the other didn't, which was used for smartphones to gain access to the network. And, it just so happened that the profile didn't require MFA!

Score!

I broke out the trusty SSL VPN Linux tool openconnect and hopped right into their internal network.

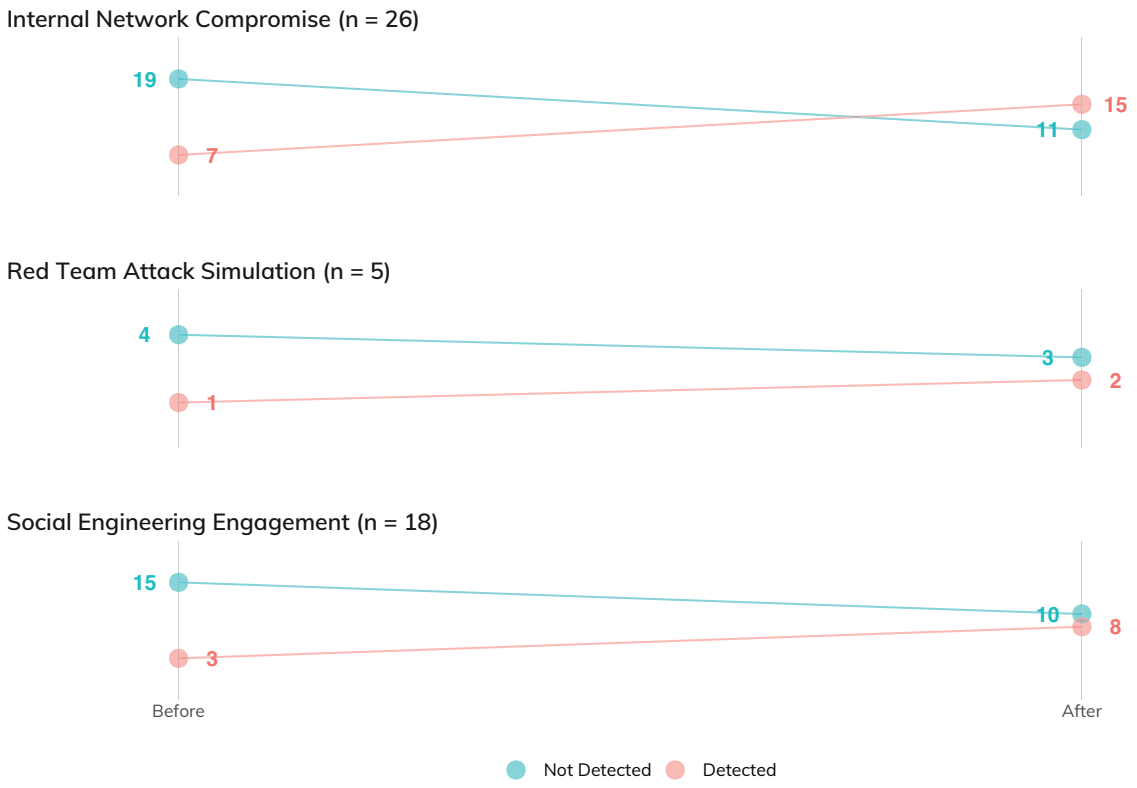


Detection, Response, and Defense

Detection and response (D&R) is a critical component of any mature security regime, since its existence recognizes that prevention technologies eventually fail in some way. That said, sometimes D&R isn't in scope for a given penetration testing engagement. While it's always in scope for red team simulations—it wouldn't be much of a simulation without it—it usually isn't a real blocker in a basic internal network compromise. D&R might (and should!) alert during an internal network compromise, but typically, the penetration tester is allowed to resume the test after getting "caught," since being stealthy and avoiding those aren't really the purpose of those engagements. However, the rate of detection across engagement types is interesting:

Figure 24: Were You Detected, Caught, or Blocked by the Blue Team, SOC, or Physical Security Before or After Internal Access?

Only includes engagements where infiltration was in scope and responses provided.



Source: Rapid7

Figure 24 indicates how often the pen tester was noticed before (on the left) and after (on the right) internal access to an in-scope asset was achieved. When D&R fires before access is obtained (red, on the left), we can be confident that D&R is doing its job as a primary control. If it fires after access obtained (red, on the right), it's doing its job as a secondary control. Although the sample size is small for red teams, we can see that usually they are undetected throughout the engagement, while standard internal network compromises are detected somewhat more often after compromise, meaning that the organization has at least some chance to notice and respond to the type of breach performed.

Fundamental Prevention

While automated D&R is a critical component of a mature IT security program, there are some fundamentals that all organizations would do well to review before their next penetration test. Broadly, these fundamentals are:

- **Review your password management strategy.** So many penetration tests end up with lists and lists of poorly chosen, human-generated passwords, regardless of whatever other fancy security controls are in place. Credential management can, and should, be a full-time function of an organization's IT security program, and the more users and services that can be moved to automatically generated, automatically rotated passwords, the better. It can be painful at first to adopt machine-controlled password management, but after a while, it will become second nature. Training and supporting users in the use of password management solutions will make them all but impervious to accidentally leaking passwords through social engineering, as well as render a purloined NTDS.dit hash file all but useless.
- **Review your patch management strategy.** The best use of an internal network assessment is to figure out where your patch management strategy is failing—what dark, cobwebby corners of your IT infrastructure aren't getting routine reviews for patches and updates. If an enterprise is relying on the users to do the right thing and click through those nag screens for updates (instead of hitting "later" again and again, forever), your penetration testers will almost certainly be overwhelmed with so many exploitable vulnerabilities that you'll only hear about the most egregious examples of where individual people have failed to keep up on updates.
- **Employ network segmentation wherever possible.** Small, manageable network segments can do worlds of good when it comes to containing an internal breach. Penetration testers spend a fair amount of time and energy not so much on getting that first shell, but in deciding where to go next. Big, flat networks can present hundreds of opportunities for lateral movement, and also tend to be difficult to manage with expensive D&R programs. The name of the game here is to bottle up attackers, both criminals and penetration testers, and make it difficult to move from asset to asset in search of systems that are both compromisable and useful launching points for the next compromise.

Sites that are lacking in one of these three fundamental information security practices can be compromised trivially by an experienced attacker, and to be honest, paying for a penetration test when one or more of these practices aren't being followed is probably a waste of everyone's time.[12]

A segmented network with reasonable patch management and machine-generated passwords for all user and service accounts is just about the most secure position to be in when the penetration tester shows up. This will ensure that the client is getting the absolute most bang for their offensive security buck. Penetration testing isn't really about testing technologies and policies; that's the job of QA and HR. Instead, it's best to think of pen testing as being about testing assumptions about those technologies and policies.

Penetration tests are almost always about making incremental improvements to existing processes and technologies, rather than pointing out that a process is totally lacking in the first place. The most impactful penetration tests are those that find that one lone system or that one forgotten network that somehow got looked over by in-house staff, and report back exactly how they found it and what they did to compromise it. In the end, penetration testing is just about the most effective way to to learn where assumptions don't match up with reality when it comes to running intensely complicated information technology, and the best penetration testers will make this the focus of any after-action report so the client can adjust their assumptions accordingly.

[12] That said, if you need a business justification for tackling one of these areas, a penetration test can go a long way to help you convince your management that time and resources should be spent!

QUESTIONS?

Email us at research@rapid7.com

RAPID7