RADIUS Accounting Considered Harmful

Andrew Lee, Selena Qiao, Maggie Yao

Abstract — The Remote Authentication Dial-In User Service (RADIUS) protocol forms a foundational component of modern-day networking infrastructure. Although RADIUS has been in use for over three decades, its accounting mechanism responsible for logging data such as session usage and billing information remains a critical component in widely-adopted infrastructure, supported by vendors like Microsoft, Cisco, Nokia, and Oracle. This paper presents a practical exploration of a cryptographic attack against RADIUS Accounting, building upon the Blast-RADIUS vulnerability that exploited flaws in RADIUS Authentication [1]. By leveraging a chosen-prefix MD5 collision, we demonstrate how an attacker with man-in-the-middle capabilities can forge a valid Accounting-Request packet without knowledge of the shared secret. Using the HashClash framework [2], we show that collisions can be computed in under eight hours when multiple gibberish attributes are used. Our work outlines the feasibility of this attack, identifies practical challenges, and proposes mitigations to strengthen the integrity of RADIUS Accounting.

I. Introduction

RADIUS is a networking protocol designed to provide authentication, authorization, and accounting (AAA) and is the gold standard for remote access network users and administrators regarding networked devices. The prevalence of RADIUS cannot be understated as it was created in 1991 but is now supported by "essentially every switch, router, access point, and VPN concentrator product sold in the last twenty years" [3]. RADIUS is widely used by major Internet Service Providers, telecommunications companies, eduroam, and many other services.

The accounting aspect of RADIUS is intended to capture data for the purposes of network monitoring and billing [4]. Therefore, large enterprises rely on it to maintain accurate accounting charge information. For instance, RADIUS Accounting is utilized in the Oracle Communications Session Border Controller (OCSBC), an industry-leading session border controller (SBC) for fixed line, mobile, and over-the-top (OTT) services [5]. Microsoft, Cisco, and Nokia all support RADIUS Accounting in their products to provide customers with robust network usage monitoring and management capabilities [1].

A. *RADIUS Accounting Attack Overview*

As aforementioned, RADIUS Accounting is designed to record information for network monitoring and billing purposes [4]. In this protocol, an end user first connects to a Network Access Server (NAS), which periodically sends Accounting-Request packets to the RADIUS server. To ensure the integrity and authenticity of each Accounting-Request, the packet includes a Response Authenticator, a MD5 hash computed over the packet header, attributes, and shared secret, in that specific order. Our attack will force the RADIUS server to log an Accounting-Request packet with a forged attribute without any knowledge of the shared secret.

Our RADIUS Accounting attack builds on the Blast-RADIUS attack described in the paper RADIUS/UDP Considered Harmful, published in June of last year [1], which targeted RADIUS Authentication. While that paper only briefly outlined a potential attack on RADIUS Accounting, we expand upon their idea. Our attack model assumes man-in-the-middle network access is possible, a plausible threat given that most RADIUS traffic is still transmitted over UDP despite longstanding security concerns.

Our attack begins when the end-user adversary triggers the NAS to generate an Accounting-Request packet containing carefully crafted gibberish attributes. Before this packet reaches the RADIUS server, the adversary intercepts and replaces it with a malicious Accounting-Request that includes a forged attribute of their choosing, along with a different set of gibberish attributes. These two versions of gibberish attributes are precomputed to produce a matching Request Authenticator using an MD5 chosen-prefix collision, allowing the RADIUS server to accept the forged packet as the server believes it to be valid.

By developing a script based on the open-source HashClash framework [2], we estimate that computing a collision using only one gibberish attribute would take at least ten days. However, when using two or more gibberish attributes, we were able to compute a collision in approximately eight hours. Including multiple attributes requires the use of a specialized technique previously developed to fix byte values in the gibberish as a result of the presence of headers within each attribute. [1].

Although we do not present a concrete attack on a specific RADIUS implementation due to limitations discussed later, we demonstrate that such an attack is indeed feasible in this paper as well as provide potential mitigations to prevent our attack.

B. Contributions

Our contributions regarding our project are as follows:

1) Conducting a practical evaluation of the RADIUS Accounting attack implementation
2) Identifying and analyzing the challenges and limitations involved in executing the attack
3) Measuring the computational time required to generate the collision on which the attack relies

II. Background

As previously mentioned, our attack builds on the RADIUS Authentication vulnerability identified in the Blast-RADIUS attack. The following sections will summarize their approach and provide an overview of RADIUS Accounting to set-up the implementation of our attack.

A. RADIUS Authentication

RADIUS operates using a client/server model, where the client forwards user credentials and connection requests to a centralized RADIUS server, which performs authentication, authorization, and accounting. In regard to authentication, the user will input their credentials to send to the RADIUS Client who will send an Access-Request to the RADIUS Server [6]. The RADIUS Server will check the credentials and send an Access-Accept or Access-Reject packet back accordingly. The RADIUS Client will then successfully allow the user to login depending on the packet received.

B. Response Authenticator

In order to determine the legitimancy of Access-Reject and Access-Accept packets, the packet will include a Response Authenticator which is computed as a MD5 hash, specifically $\text{MD5}(\text{Code}||\text{ID}||\text{Len}||\text{ReqAuth}||\text{Attributes}||\text{Secret})$. With the exception of the ReqAuth and the Secret, all the information is copied from the Access-Reject and Access-Accept packet. The ReqAuth is taken from the original Access-Request packet which represents a random 16-byte nonce generated for each authentication session attempt. The Secret is a shared secret previously agreed upon between the RADIUS client and server.

C. MD5 Collisions

The MD5 protocol was designed in 1991 through utilizing Merkle-Damgard to iteratively process the input in blocks of 512 bits to compress to a final 128-bit output [7]. Although MD5 was shown to be insecure in 2004 when researchers demonstrated a practical collision using a birthday attack [8], it continues to be used in the RADIUS protocol due to its longstanding integration and widespread legacy deployment.

Further developments led to the MD5 chosen-prefix attack in which given fixed prefixes P_1 and P_2, it's possible to produce gibberish G_1 and G_2 such that $\text{MD5}(P_1||G_1) = \text{MD5}(P_2||G_2)$ [9]. The interesting part of the attack is that anything that comes after the gibberish, as long as it's identical, will continue to be a collision. Therefore, $\text{MD5}(P_1||G_1||S)$ and $\text{MD5}(P_2||G_2||S)$ will continue to produce a collision.

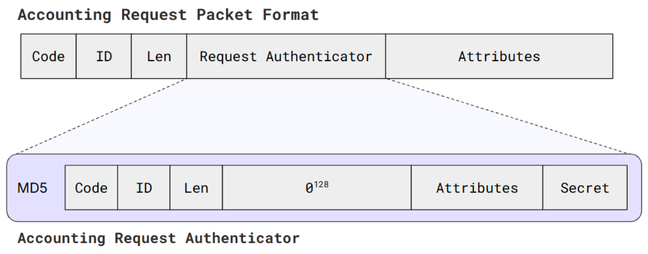

Fig. 1: A RADIUS accounting packet and Request Authenticator

D. Blast-RADIUS

The Blast-RADIUS attack enables an adversary to bypass RADIUS Authentication and gain access using invalid credentials, potentially even as an administrator, as illustrated in Figure 2. As mentioned in the previous section, MD5 has long been known to be insecure, and the crux of the attack is a MD5 chosen-prefix collision regarding the Request Authenticators of the Access-Reject and Access-Accept packets through a man-in-the-middle attack.

The adversary will first input invalid credentials to the RADIUS client who then sends an Access-Request to the RADIUS server. The adversary will then intercept the Access-Request and compute RejectGib and AcceptGib such that $\text{MD5}(\text{AcceptPrefix} \,\|\, \text{AcceptGib}) = \text{MD5}(\text{RejectPrefix} \,\|\, \text{RejectGib})$ to obtain identical Response Authenticators. As demonstrated above, as long as the shared secret comes after the gibberish, the two computed Response Authenticators will continue to be identical. To allow for enough space for the chosen-prefix collision, the gibberish is computed through the addition of Proxy-State attributes. A Proxy-State attribute includes a one-byte code, a one-byte length field, and a data string of up to 253 arbitrary bytes. Proxy-State attribute were chosen due to the property that they “MUST be copied unmodified and in order into the response packet” [6]. It is important that this computation has to be done online due to the 16-byte random nonce generated for each authentication session.

The adversary will then send an Access-Request packet with the appended RejectGib. Then when the RADIUS server rejects the invalid credentials, it’ll send an Access-Reject packet with the RejectGib appended. The adversary will intercept the transmission once more and instead send the RADIUS client an Access-Accept packet with the AcceptGib, copying over the computed Request Authenticator found in the Access-Reject packet. As a result, the RADIUS client will accept the Access-Accept packet as valid because the Request Authenticator appears to be correct and the adversary can successfully authenticate.

E. RADIUS Accounting

In comparison to RADIUS Authentication, RADIUS Accounting is used for network monitoring, usage statistics, and billing users accurately [10]. The process begins when a user connects, and the NAS sends an ”Accounting Start” Accounting-Request packet to the RADIUS server with session details like user ID and IP address. During the session, the NAS may send ”Interim Update” Accounting-Request packets with current usage statistics such as the username, number of input packets, session duration, etc. When the session terminates, an ”Accounting Stop” Accounting-Request packet is sent with final data on time, usage, and disconnection reason. The client will send Accounting-Request packets until the RADIUS server acknowledges receipt with an Accounting-Response packet and the server will store the relevant packets in its log.

Similarly to RADIUS Authentication, the Accounting-Request packet will also include a Request Authenticator to prevent forgery of the packets. The only difference is that instead of a random 16-byte nonce for the session, 0^{128} in an attempt to prevent attacks like the one described in this paper. The MD5 hash therefore looks like $\text{MD5}(\text{Code} \,\|\, \text{ID} \,\|\, \text{Len} \,\|\, 0^{128} \,\|\, \text{Attributes} \,\|\, \text{Secret})$ as demonstrated in Figure 1. If the RADIUS client detects that the Response Authenticator is invalid, it will simply discard the packet.

F. FreeRADIUS

The implementation that we base our evaluation upon is called FreeRADIUS [11]. Almost if not all RADIUS servers are based on one of the following five implementations: FreeRADIUS, Radiator,

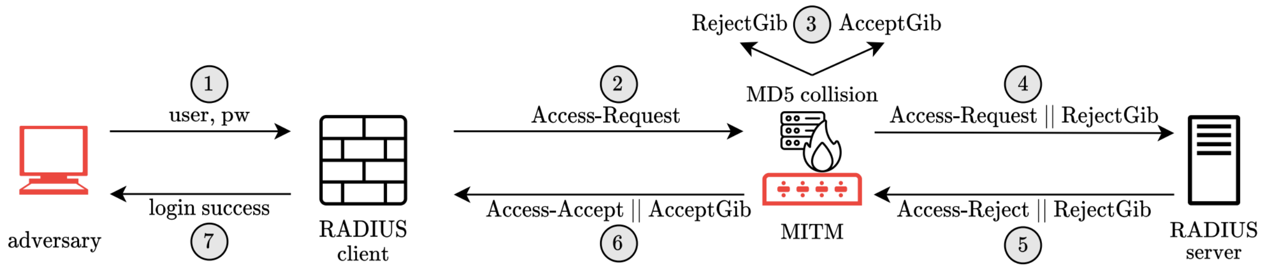

Fig. 2: The attack flow illustrated in Blast-RADIUS, with the image sourced from their paper. The adversary enters invalid credentials (1) and then mounts a man-in-the-middle attack (2) in which Response Authenticator collision gibberish is computed online (3) and inserted in the Access-Request (4). Although the server rejects the invalid credentials (5), the adversary intercepts the Access-Reject, replacing it with an Access-Accept containing the forged gibberish (6). By exploiting the same Response Authenticator, the client unknowingly authenticates the adversary due to the collision (7).

Cisco, Microsoft, and Nokia [1]. We chose to evaluate the FreeRADIUS implementation because it is the most widely used of the five RADIUS implementations and is open source.

III. Radius Accounting Attack

Because the Response Authenticator in a RADIUS accounting packet is essentially the same format as that of an Access-Accept or Access-Reject packet, these packets are susceptible to the same type of forgery attack. In this section, we describe the general attack flow, practical challenges of extending this attack identified in the BlastRADIUS paper, and how these challenges can be overcome.

A. Attack Flow

In our attack as demonstrated in Figure 3, an attacker with control over the network can forge a RADIUS accounting request containing invalid logs. The attack flow differs from that of the original attack because the objective is to forge a request and not a response. The attacker must compute two sets of collision gibberish, one for a legitimate accounting request, and one for the forged accounting request, then replace the legitimate request with the forged one. We then ignore the responses from the RADIUS server; they are no longer important to the attack.

A sketch of the full attack is as follows. First, the attacker computes a pair of collision bytes between the legitimate and forged accounting requests. Then, the attacker triggers a RADIUS client such that it generates the legitimate accounting packet, but intercepts this packet on the network and replaces it with the forged accounting request with the same authenticator. The RADIUS server will recompute the authenticator with the forged packet fields and validate the integrity of the message.

B. Timeout

One benefit of this modified attack flow is that we are not subject to the same timeout as the original BlastRADIUS attack. A big challenge in the original paper was computing an MD5 collision within a few minutes in order to forge an Access-Accept before the authentication timeout. This required many machines to work in parallel, as well as modifying collision-finding to operate on distributed compute.

There is no restriction on the time at which an accounting request can be sent. (However, most implementations of RADIUS will include the timestamp of the request as a request attribute.) This greatly reduces the amount of computational power necessary to complete this attack, and an attacker can take a longer time to compute a collision while still being able to complete the attack successfully.

C. Request Triggering

In this attack flow, the attacker must trigger the client to send a legitimate RADIUS accounting request. This is in contrast to the original attack, in which the attack focuses on the response. This made it possible to insert arbitrary collision bytes

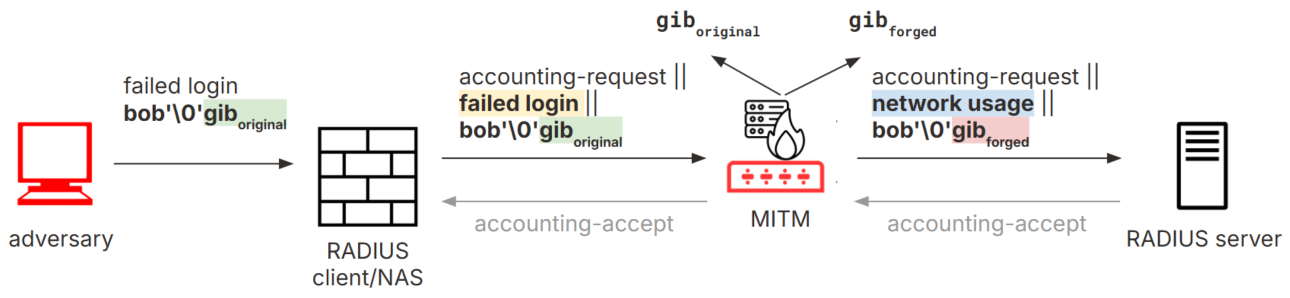

Fig. 3: The flow of our attack. The adversary precomputes an MD5 collision then triggers a failed login, which generates a legitimate accounting request. The adversary then intercepts that packet and sends the forged packet.

in the Proxy-State attribute because these bytes may be ignored by the server upon receipt but must be included in the response. As stated in the original paper, this property cannot be exploited by the RADIUS Accounting attack because this attack focuses on forging a request, and not a response.

Instead, any attributes that contain collision bits must be populated as a result of triggering a RADIUS client to send a legitimate request – this is so that we can obtain the Request Authenticator of the packet, which contains the shared secret as part of its pre-image.

This significantly limits the types RADIUS attributes that we can populate with gibberish. Proxy-State is no longer a viable option because no reasonable implementation of RADIUS would allow a user to populate the attribute with gibberish. One viable option identified by the paper is to use the username field – this can be very reasonably triggered by attempting a login, for example.

We verify in our evaluation that the username field likely provides enough space (253 bytes) to compute a collision, especially considering that the collision computation is not subject to a timeout constraint.

There are several other RADIUS accounting attributes that may be candidates for storing collision gibberish because they can store up to 253 bytes in their value (any string type attribute is in theory a candidate). These attributes can be combined to form a collision by fixing header bytes, as described in the next section. However, how or whether these fields are populated will be highly dependent on the implementation and likely hard to manipulate.

The authors suggest that the attacker can either predict or cause the message the RADIUS client will send, then compute a collision based on this prediction. However, purely anticipating a RADIUS request from the client (like a periodic update) will likely not work. This is because the attack requires including collision gibberish in the original request, which cannot happen by passively predicting packets.

D. *Packet Prediction*

A related practical challenge of this attack identified by the original authors is packet prediction. Recall that in this attack flow, the collision gibberish must be computed before the original accounting packet is actually sent. This implies that a prefix of the packet content must be known prior to the packet actually being sent.

This prediction may be complicated, as packet attributes include automatically generated timestamps and session IDs, which are typically implementation-dependent. However, as we will note in the next section, these attributes need not be predicted.

In the case where these attributes must be predicted, the attack remains feasible, though more challenging. The attacker would first need to snoop an active RADIUS session to capture the session ID. Next, they would compute a collision using the two prefixes, based on a preselected timestamp in the future. The legitimate request would then need to be triggered precisely at that time. Since timestamps are measured in seconds, accurate timing is reasonably achievable. This version of the attack would also depend on the session timeout to ensure the same session ID remains valid; however, this timeout is typically longer than the authentication timeout used in the original attack.

E. Request Triggering

The last practical challenge identified by the original authors is that only attributes that come before the collision gibberish can be forged. This is because once an MD5 collision is generated, any bytes that come after must be identical in order for the two messages to continue colliding (otherwise the state diverges again).

We have found that the ordering of these attributes is not specified by the RADIUS protocol and is instead implementation dependent. In FreeRADIUS, the username attribute, which contains the collision gibberish, comes after the attributes we want to forge; therefore, the issue of attribute ordering is avoidable.

Another feature of FreeRADIUS is that it always places the session ID and the timestamp after all other client-generated attributes. Recall that these are the two attributes that make packets hard to predict. However, if these attributes come after the collision gibberish, they need not be included in the predicted prefix at all. Instead, the attacker can merely copy the exact same timestamp and session ID off of the legitimate packet and thus maintain the MD5 collision without any extra work.

F. Username Validity

One issue with inserting gibberish into the username is that it makes the username invalid; this means that it may be harder to forge meaningful logs. However, it is possible that there exists some vulnerable implementation that parses usernames until a null byte, yet when computing Request Authenticators includes every byte of the attribute. The paper references Marlinspike and Kaminsky's null byte attack against SSL/TLS as an example of this type of attack in the past [12].

While this would mean that this attack would only work against such vulnerable implementations, we note that it is still possible to forge meaningful requests that are not necessarily attached to a user. In addition, as mentioned before, it may be possible to insert gibberish into attributes other than the username, which removes this limitation.

G. Final Attack

An example of a practically feasible attack on RADIUS accounting is as follows. The attacker's goal will be to forge accounting requests for some user Bob such that the RADIUS server believes Bob network's usage is unusually high by forging the Acct-Input-Gigawords field, a field essentially used to track the amount of data received by the user from the network, increasing the billing cost significantly. First, the attacker determines a valid trigger for the RADIUS client that allows us to insert collision gibberish. For instance, the attacker could trigger a failed login for Bob by attempting to authenticate as Bob with the wrong password, which will generate a RADIUS accounting request from the RADIUS client.

The attacker constructs the packet prefix for the packet that would be sent, then constructs the prefix for the desired packet with a large Acct-Input-Gigawords value. The two prefixes should both include `Bob\0` in the username field so that the request is attached to the user Bob. Then, he computes a four-block MD5 collision of the two prefixes and inserts the rest of the gibberish in the remaining username attribute space.

Finally, the attacker triggers the failed login, intercepts the request from the client, and replaces it with the forged request.

IV. MD5 Collision Computation

To generate MD5 collisions for a practical application, we used HashClash, an open-source framework designed for constructing collisions against MD5 [2]. The process exploits MD5's structural vulnerabilities via a two-phase approach that leverages a variant of the birthday attack followed by differential path construction.

The first step is to generate a near collision, a pair of message blocks that produce internal MD5 states that differ in only a few bits. This phase relies on the classic birthday paradox: by generating and hashing a large number of message prefixes with chosen differences, HashClash attempts to find pairs whose MD5 state differences fall within a narrow subspace that makes the second step easier. This step is computationally intensive but parallelizable.

Once a near collision is found, HashClash transitions to the path-finding phase. Here, it searches for a valid differential path that propagates the small initial differences through the MD5 compression function, ultimately resulting in an actual hash collision. This involves adding a carefully crafted sequence of additional message blocks.

There's a tradeoff between the time spent finding a near-collision, the time spent finding a collision path, and the number of message blocks required. Finding a shorter path may take significantly longer computational time, while accepting longer paths (i.e., more added blocks) makes the search easier but increases the final payload size.

An important constraint when extending the collision chain is the format of the message blocks. In the specific context we were working in, every group of four blocks after the initial collision needed to start with two fixed bytes of our choosing. This was due to protocol or file format requirements (e.g., attribute headers that had to conform to certain patterns). As a result, each subsequent collision extension had to satisfy not just the MD5 differential requirements, but also these structural prefix constraints.

We developed a script in HashClash to fix these bytes, adapting the framework to meet the specific constraints of our application. However, imposing prefix constraints on blocks narrows the space of valid solutions, making path-finding more complex.

V. Evaluation

A. Attack Computational Time

The computational effort required to generate a successful MD5 chosen-prefix collision depends heavily on the amount of message space available for injecting controllable data. In our setting, this space corresponds to the number and size of attribute fields we can exploit. Each attribute permits up to 253 bytes of arbitrary data, equivalent to 4 MD5 message blocks. Thus, the total number of attributes we can use directly translates into how many blocks we can devote to satisfying the collision path requirements. For all of our evaluations, we performed them using a single machine.

If we are limited to only a single attribute—such as a username field—this restricts us to just 4 blocks after the shared prefix. Under this constraint, the space of possible message differences and differential paths narrows significantly, making the search for a valid collision far more time-consuming. Based on our estimates and practical trials, generating a chosen-prefix collision under these constraints would take at least 10 days on a single machine, and likely significantly more depending on hardware and optimization.

Nevertheless, previous experiments have shown that with increased computational power, it is possible to compute a chosen-prefix collision within a single block. In 2009, Stevens et al. demonstrated the feasibility of generating a chosen-prefix MD5 collision to forge an adversarial TLS certificate authority in just 204 bytes [13]. This was achieved by leveraging 200 Sony PlayStation 3 consoles running in parallel, completing the task in approximately 28 hours of computation.

In contrast, if two attributes are available—yielding 8 controllable blocks—we observed a dramatic improvement in performance. In this case, we successfully computed a chosen-prefix collision in approximately 8 hours. These results highlight the sharp computational tradeoff between available payload space and collision generation time.

B. Limitations

While our attack demonstrates the feasibility of forging RADIUS accounting requests using chosen-prefix MD5 collisions, it is subject to several practical limitations that constrain its applicability in real-world scenarios.

To obtain the Request Authenticator and ensure valid packet formatting, the attacker must trigger a legitimate RADIUS accounting request that includes the collision gibberish. This limits the set of attributes into which gibberish can be inserted. So far, we have found the username field to be the only viable candidate—users can often supply arbitrary usernames, and the field typically supports up to 253 bytes. Other attributes like Proxy-State are either server-controlled or impractical to manipulate in a meaningful way.

The attacker must accurately predict the content of the accounting request prior to its generation, including all header and attribute values up to the collision point. This includes fields like timestamps and session IDs, which are often implementation-dependent and may vary across deployments. While some of these fields may come after the collision blocks and can therefore be ignored, this ordering is not guaranteed and must be verified on a per-implementation basis.

MD5 collisions require both message prefixes to diverge and then reconverge. Therefore, any forged attributes must appear before the collision blocks. Attributes that come after the collision blocks must be identical across both the legitimate and forged messages, which severely limits forgery capabilities. Since RADIUS does not specify a canonical attribute ordering, this again depends on implementation behavior.

Each attribute can hold at most 253 bytes, equivalent to four MD5 blocks. Since MD5 chosen-prefix collisions require multiple such blocks to construct a valid differential path, the number of available attributes directly limits the feasibility and speed of the attack. At this point, we have only found one attribute could be reasonably filled with gibberish, significantly increasing the computational cost of finding a collision.

Taken together, these limitations mean that while the attack is feasible under controlled conditions—especially against certain configurations like FreeRADIUS—it may not generalize to all RADIUS deployments without significant customization or inside knowledge of the target system's behavior.

VI. Mitigations

The following subsections will discuss potential avenues regarding mitigating the RADIUS Accounting attack. These strategies aim to enhance the protocol's security and safeguard it against potential future attacks as well.

A. Non-Mitigation: Increasing Secret Length

Although increasing the length of the secret is a common method for strengthening the security of other schemes, it does not affect our attack. This is because the length of the secret is irrelevant—our attack relies on the fact that the secret always follows the gibberish.

B. Short-Term Mitigation: Inspecting Logs

The simplest strategy does not attempt to stop the attack but rather to detect it. Since all information related to the accounting session is stored on the RADIUS server, typically for later billing inspection, one can simply review or have a script to filter the logs to identify suspicious activity. This is because our current implementation will generate gibberish in the literal sense to find the MD5 collision. However, it remains plausible that future advancements in MD5 collision techniques could produce intelligible gibberish, undermining the effectiveness of this mitigation strategy.

C. Non-Mitigation: Deprecating MD5

While the most obvious mitigation may be to eliminate MD5 entirely in favor of secure hash functions like SHA-256, such measures are largely incompatible with existing implementations due to the difficulty of altering RADIUS's cryptographic practices. This mitigation would be essentially utilizing an entirely new protocol. The below mitigations are more effective while maintaining the RADIUS protocol.

D. Short-Term Mitigation: Message-Authenticator

The recommended short-term mitigation that was implemented as a result of the RADIUS/UDP Considered Harmful paper is to require a Message-Authenticator as the first attribute [1]. The Message-Authenticator as described in RFC 2869 is an optional attribute that computes an HMAC-MD5 over the entirety of the packet [14]. Currently, HMAC-MD5 is considered secure, unlike MD5, assuming the underlying pseudorandom functions are themselves secure [15]. Therefore, it will be impossible to run our attack on the modified protocol as the HMAC-MD5 attribute will be effectively random.

The crucial requirement is that the RADIUS server must always include the Message-Authenticator, and the RADIUS client must always verify it. As a result of the aforementioned paper, all known RADIUS implementations have incorporated the above mitigation into their protocols, preventing our attack [16]. However, the mitigation remains vulnerable to a potential downgrade attack, the feasibility of which we leave to future considerations.

E. Long-Term Mitigation: Transport Security

Through transmitting RADIUS data over an encrypted and authenticated channel with modern cryptographic guarantees, the demonstrated attack can be more effectively prevented. While migrating to DTLS/TCP and TLS entails substantial infrastructure changes, it provides the most robust and reliable protection against both RADIUS Authentication and Accounting attacks, and is strongly recommended for future deployments.

VII. Conclusion and Future Work

In this paper, we present an attack that allows adversaries to forge RADIUS Accounting logs through a man-in-the-middle approach that leverages an MD5 chosen-prefix collision. Our evaluation demonstrates that while the attack is practical, it faces several limitations. Specifically, successful execution depends on addressing challenges related to timing, attribute ordering, and variations in implementation behavior. We also analyze the computational effort required to generate the necessary collision, providing a clearer picture of the attack's feasibility in real-world scenarios.

For future work, one potential direction is to evaluate the feasibility of a downgrade attack, especially in environments where the Message-Authenticator mitigation is currently deployed. It is also critical to begin migrating RADIUS to more secure transport protocols such as DTLS/TCP and TLS, and to assess its security in those contexts. Overall, our project highlights the importance of continually assessing the security of real-world deployments, regardless of how long they have been in use, as longevity does not guarantee continued security.

VIII. Contributions

All authors contributed equally to brainstorming project ideas, reading relevant papers, creating the foundations of the attack, developing the script for computing the collision, evaluating the limitations and mitigations, and writing up the final project report.

IX. Acknowledgements

Thank you to our TA Katarina Cheng as well as Professor Yael Kalai and Professor Henry Corrigan-Gibbs for their advice and teachings. Special thanks to Marc Stevens and Miro Haller of the Blast Radius Attack Team for all their help with making our project possible!

References

[1] S. Goldberg, M. Haller, N. Heninger, M. Milano, D. Shumow, M. Stevens, and A. Suhl, "RADIUS/UDP considered harmful," in *Proceedings of the 33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA, USA: USENIX Association, Aug. 2024, pp. 7429–7446. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/goldberg

[2] M. Stevens, "Project hashclash - md5 and sha1 cryptanalytic toolbox," https://github.com/cr-marcstevens/hashclash, 2009, accessed: 2025-05-11.

[3] A. DeKok, "Radius and md5 collision attacks," 2024. [Online]. Available: https://networkradius.com/assets/pdf/radius_and_md5_collisions.pdf

[4] C. Rigney, "Radius accounting," RFC 2866, June 2000, https://datatracker.ietf.org/doc/html/rfc2866.

[5] Oracle Corporation, "Configuring accounting," 2024, accessed: 2024-05-12. [Online]. Available: https://docs.oracle.com/cd/E95618_01/html/sbc_scz810_accounting/GUID-5D866DB4-8537-4CCC-A1EB-1F649925E9A3.htm#Configuring-Accounting

[6] C. Rigney, S. Willens, A. Rubens, and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)," https://datatracker.ietf.org/doc/html/rfc2865, 2000, rFC 2865.

[7] R. L. Rivest, "The md5 message-digest algorithm," RFC 1321, April 1992, request for Comments 1321. [Online]. Available: https://www.rfc-editor.org/rfc/rfc1321

[8] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Advances in Cryptology - EUROCRYPT 2005*, ser. Lecture Notes in Computer Science, vol. 3494. Springer, 2005, pp. 19–35.

[9] M. Stevens, A. K. Lenstra, and B. de Weger, "Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities," in *Advances in Cryptology - EUROCRYPT 2007*, ser. Lecture Notes in Computer Science, vol. 4515. Springer, 2007, pp. 1–22.

[10] NetworkRADIUS, "How does radius accounting work?" https://networkradius.com/articles/2019/06/05/how-does-radius-accounting-work.html, June 2019, accessed: 2025-05-11.

[11] The FreeRADIUS Project, "Freeradius," https://www.freeradius.org/, 2025, accessed: 2025-05-11.

[12] M. Marlinspike, “Null prefix attacks against ssl/tls certificates,” https://www.blackhat.com/presentations/bh-usa-09/MARLINSPIKE/BHUSA09-Marlinspike-DefeatSSL-PAPER1.pdf, July 2009, black Hat USA.

[13] M. Stevens, A. Sotirov, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger, “Short chosen-prefix collisions for md5 and the creation of a rogue ca certificate,” in *CRYPTO*, ser. Lecture Notes in Computer Science, vol. 5677. Springer, 2009, pp. 55–69.

[14] W. Willats, C. Rigney, and P. R. Calhoun, “Radius extensions,” RFC 2869, 2000, https://www.rfc-editor.org/rfc/rfc2869.

[15] M. Bellare, “New proofs for nmac and hmac: Security without collision resistance,” *Journal of Cryptology*, vol. 28, no. 4, pp. 844–878, Oct. 2015. [Online]. Available: https://doi.org/10.1007/s00145-014-9187-1

[16] S. Goldberg, M. Haller, N. Heninger, M. Milano, D. Shumow, M. Stevens, and A. Suhl, “Blast-radius,” https://www.blastradius.fail/, 2024, accessed: 2025-05-11.