**Slide 1 (Front page):**

Good morning. My name is Louis Dodge, and I'll be presenting my Master's research proposal on detecting privilege escalation paths in cloud environments before code is merged. Let's begin.

**Slide 2 (Overview):**

To start, just a quick overview of what we’re going to be talking about today

First I’m going to go over what the research problem here is, as well as the 3 research questions I specifically intend to address

Then I’ll touch briefly on key literature for this research project, followed by an outline of my methodology and ethical considerations,

Before finishing with a proposed timeline and outline of artefacts to be created during the project.

**Slide 3 (Research Problem):**

In computing environments, excessive permissions remain a key cause of compromise.

When it comes to tooling for identifying infrastructure security risks, organisations rely largely on static code analysis pre deployment, and live environment scanning post deployment.

These current approaches bring value, but both have significant limitations.

Static analysis of infrastructure code operates without awareness of the live environment. As a result, it can’t reliably compute effective permissions or reachability - especially in hybrid estates where many resources are not managed as code or have drifted post-apply.

Post-deployment scanning, meanwhile, is reactive. It detects escalation paths only after they exist in a live environment, creating an exposure window from introduction to identification and remediation.

What’s missing is a pre-merge capability that reasons over planned changes plus current live state to identify prospective privilege-escalation paths before they’re introduced. My work targets that gap.

**Slide 4 (Research Question 1):**

I have three research questions I intend to address: The first is foundational: can my graph-based approach reliably detect privilege escalation paths pre-merge? I need to establish that the method actually works before comparing it to anything else. This answers: is the approach viable?

**Slide 5 (Research Question 2):**

The second question compares detection performance. How do detection rates differ and in which direction? Graph reasoning might perform better, worse, or equivalently compared to static scanning. Critically, I'll report results stratified by pattern type. We might find graph reasoning helps for multi-hop patterns but not for direct over-privilege cases, for example.

**Slide 6 (Research Question 3):**

The third question compares false positive rates. Both detection rate AND false positive rate matter operationally, as high false positive rates can lead to genuine issues being ignored. I'll characterize both dimensions to assess the practical tradeoffs.

This framing allows the data to tell us where graph reasoning provides value, if at all.

**Slide 7(Aims and Objectives):**

The aim is to design and rigorously evaluate this graph-aware approach for pre-deployment privilege escalation detection in Azure subscription-scope environments.

To that end, I have four key objectives. First, to precisely define what counts as a privilege escalation path and specify the evaluation protocol.

Second, generate a test corpus with validated ground truth. I'll create 160 representative PRs across different pattern types and establish objective ground truth through an oracle process. I'll explain the generation and validation methodology in detail shortly.

Third, to implement a working prototype that fuses Terraform plans with live Azure state.

Finally, to evaluate the performance of the graph approach, including providing stratified results by pattern type to understand where each approach excels.

**Slide 8 (Key Literature):**

Let me position this work against existing research.

First, graph-based attack path analysis. Elmiger and colleagues published in the International Journal of Information Security in 2023, demonstrating how graph-based analysis can systematically reduce attack paths in Azure enterprise environments using Neo4j. Their work built on industry tools like BloodHound and AzureHound, which practitioners use for privilege path enumeration in Active Directory and Azure. These approaches model relationships- group membership, role assignments, resource ownership- and find multi-hop escalation chains. But they're fundamentally post-deployment.

Second, Infrastructure-as-Code security. Rahman and colleagues at ICSE 2019 conducted an empirical study identifying seven common security smells in over 15,000 IaC scripts. They found issues like hard-coded credentials, admin-by-default configurations, and overly permissive settings. Critically, their static analysis detected resource-level misconfigurations but not privilege escalation paths that emerge from relationships between resources. Verdet et al.'s 2023 study of Terraform projects similarly found that static checks catch individual policy violations but not compound risks. Tools like Checkov enforce these checks early, but operate without environmental context.

**Slide 9 (Tooling Context):**

Here’s a quick overview of the main tooling I’ll be using for my research. Terraform is the infrastructure-as-code system I use to get a plan - a machine-readable diff of changes Terraform will make to infrastructure should the plan be deployed.

AzureHound is the ingestor that snapshots the live Azure/Entra graph - roles, ownership, secret access - into BloodHound’s model of traversable, abusable edges.

I use Terraform for the plan delta and AzureHound for the live-state graph, then combine them pre-merge. I’m not evaluating these tools per se; the study asks whether pre-merge fusion can predict the post-apply attack paths. If another IaC or ingestor provided the same artifacts, the method would be unchanged.

**Slide 10 (Methodology – Ground Truth Oracle):**

Establishing ground truth is critical. We need an oracle, an authoritative reference that tells us which attack paths actually exist.

First, what counts as an attack path? To determine this, I start with a known baseline environment and apply the Terraform changes from each PR in an isolated, test Azure tenant. As Azure has eventual consistency and changes take time to propagate, I then wait 20 minutes for convergence. I then snapshot the live environment with both AzureHound and Wiz, which is another tool that can similarly map out graph relationships.

**Slide 11 (Methodology – Ground Truth Oracle continued):**

Paths detected by both tools become high-confidence ground truth. In ambiguous cases where the two tools disagree, I take a conservative approach and exclude that PR. The goal of performing cross-tool validation is to prevent circularity issues, whereby ground truth is set solely by the same graph tool I’ll use pre-merge. Rather than relying on a single tool's edge definitions; I'm establishing consensus between independent detection methods.

To maintain quality, my set of ground truth PRs will include negative controls (meaning PRs without any attack paths present), pinned tooling versions for reproducibility and I’ll report on inter-tool agreement rate as a quality metric.

One small limitation is that I've scoped my study to exclude DataActions, which can require multi-hour propagation. However, as you’ll see on the next slide, our test corpus will still cover the vast majority of common privilege escalation attack patterns in Azure.

**Slide 12 (Approach at a glance):**

Step one: parse the Terraform plan JSON, extracting identity-relevant changes (e.g. new service principals, role assignments, group memberships).

Step two: snapshot the current Azure environment using AzureHound. This is crucial because without current state, we can't detect interactions with existing privileges.

Step three: merge these graphs. We apply planned changes to current state to construct what the environment will look like post-change. This will require careful handling of Terraform's lifecycle semantics, as at points new resources may be created before others are destroyed and resources can be created and assigned permissions simultaneously.

Step four: traverse this projected graph using a library of privilege escalation edges , looking for patterns like 'user member of group with Owner role on subscription.’

Step five: output detected paths and an audit bundle with versions and evidence hashes for reproducibility.

For comparison, I run three static IaC tools on each PR. Both approaches answer: 'Should this PR be flagged?' That's where I compare them.

**Slide 13 (Methodology – Test Corpus):**

The test corpus consists of 160 PRs categorised by different privilege escalation patterns.

Sample size is driven by two main factors. First, the central limit theorem begins to apply at samples of 30. Given our test categories have a minimum of 40 PRs each, I will be able to perform meaningful stratified comparisons. Second, feasibility constraints. Given time limitations, it becomes impractical to scale sample size much further than 160 PRs.

To prevent bias in PRs, an independent researcher will generate them from detailed specifications without knowledge of my detection algorithms or static tool rules. As part of ground truth establishment, each PR will be validated to confirm it produces the intended privilege escalation behavior.

Once validated, two independent researchers will classify each PR by pattern type based on its design intent as specified in the original specification. Cohen's kappa will be calculated to assess inter-rater reliability. Our target is κ ≥ 0.80, indicating substantial agreement. For PRs where researchers initially disagree, they will discuss and reach consensus through structured deliberation. Final classification for each PR will be documented along with whether it required deliberation.

**Slide 14 (Methodology – Evaluation Design):**

Let me explain how the evaluation answers each research question.

For each of the 160 PRs, both methods analyze the pull request. The graph approach parses the Terraform plan, merges it with a live Azure snapshot, and traverses the projected graph to detect privilege escalation paths. Static tools scan the Terraform code using their policy rules. Each method outputs a binary decision: flag or don't flag. I compare these predictions to ground truth from the oracle, creating a confusion matrix for each method. From this, I answer three questions.

RQ1: Can the graph approach reliably detect paths? I’ll report the graph approaches’ performance in isolation, recording true positive rate, false positive rate, precision, and F1 score, all with 95% confidence intervals. This characterizes the graph approach in isolation, answering whether it can reliably detect paths before comparing it to alternatives

RQ2: How do detection rates differ? I calculate the difference in true positive rates between methods with confidence intervals and use McNemar's test to assess statistical significance. McNemar's is appropriate because the same PRs are evaluated by both methods, meaning this is paired data. To show where graph reasoning helps, where static analysis suffices, and where neither works well, I'll stratify the results by pattern type.

RQ3: How do false positive rates compare? I compare FPR on the 40 negative controls using the same paired approach. This matters because improved detection doesn't help if false alarms make the tool unusable.

**Slide 15 (Timeline):**

The timeline is 16 weeks across three phases.

Phase one establishes the test corpus. In Week 2, I pilot the oracle. Testing Azure convergence and validating that cross-tool comparison works. Week 3, I write detailed pattern specifications. Week 4, an independent researcher generates 160 PRs from these specs, and we validate each one immediately to confirm it creates the intended privilege escalation. Week 5, two researchers independently classify all PRs by pattern type.

Phase two is building the tools. Weeks 6 through 11, I implement the graph fusion engine and detection algorithms.

Phase three is evaluation. Week 13, I run the oracle on all 160 PRs to establish ground truth using cross-tool validation. Weeks 14-15, locked evaluation. Both detection methods analyze all PRs with no further changes allowed. Week 16, analysis.

The Week 2 pilot is critical for de-risking the entire approach. If results are negative, that's still valuable. It shows where added complexity isn't justified.

**Slide 16 (Research Ethics and Integrity):**

Bringing this to a close, finally I'll address research integrity and ethical considerations.

The primary commitments relate to methodological transparency and addressing bias. The analysis plan is documented in this thesis proposal before data collection. I'll report all statistical tests, including non-significant results. For the stratified analysis across five pattern types, I'll report all findings, not selectively choose favourable ones. Negative results will be published.

To address experimenter bias, the corpus is generated by an independent researcher who doesn't know my detection algorithms. Two researchers independently classify pattern types. Ground truth uses cross-tool validation rather than relying on a single tool.

For reproducibility, the complete methodology, all tool versions, and analysis procedures will be documented so others can replicate this work.

The research uses isolated synthetic Azure tenants following standard security research practices, with university ethics approval.