FOLIOVISTA BOOKS FREE SAMPLE

Operational Bug Bounty Fieldwork

This free sample chapter introduces the scope discipline and operating rules that should come before recon, Burp, scripts, or any other testing activity.

Format: Free Sample Chapter **Included: Chapter 01**

Paid chapters later: Recon + Burp

This public sample includes only **Scope and Operating Rules**. The later chapters, **Recon to Testable Hypotheses** and **Burp Baseline and Account Comparison**, can be published later as paid content.

Main book page **Contact FolioVista Books**

01 - Scope And Operating Rules

Objective

Before touching Burp, Nmap, a browser profile, or a script, you need operating rules. This chapter defines the rules that keep bug bounty work useful, repeatable, reportable, and authorized.

The goal is simple:

`Only test what you are allowed to test, only collect evidence you need, and only report issues that demonstrate real security impact.`

This sounds basic, but most weak bug bounty work fails here. The researcher starts with tools instead of scope. They chase noisy endpoints, rotate IPs, test third-party systems, disclose too much, or submit reports with no tangible risk. Good fieldwork starts with discipline.

The First Rule: Permission Comes Before Testing

Bug bounty work is not permission to test the whole internet. It is permission to test specific assets, under specific rules, for specific vulnerability classes, with specific limits.

Before testing anything, answer these questions:

prohibited?

identity?

- Is this asset explicitly in scope?
- Is this type of testing allowed?
- Is authentication required or prohibited?
- Are automated tools allowed?
- Are rate limits or request-volume limits stated?
- Are denial-of-service, social engineering, phishing, spam, or physical tests
- Are third-party services excluded?
- Does the program require a special header, account, email alias, or testing
- Does the program restrict public disclosure?

If you cannot answer these questions, you are not ready to test.

Scope Is A Contract

Treat scope like a contract, not a suggestion.

An asset can look interesting and still be out of scope. A public API documentation page can exist and still not give you permission to test the API. A subdomain can belong to the brand and still be operated by a third party. A bug can be technically real and still be non-reportable if you found it outside the allowed boundaries.

For each target, create a short scope card:

```
Program:
Asset:
Asset type:
Allowed testing:
Disallowed testing:
Authentication requirements:
Rate limits:
Special headers or researcher identity:
Disclosure policy:
Data handling notes:
Stop conditions:
```

Do not start hunting until this card is complete.

Public Does Not Mean Authorized

One of the easiest mistakes is confusing public visibility with authorization.

Examples:

- A public endpoint is visible in JavaScript.
- A public documentation page lists API routes.
- A sitemap exposes paths.
- A robots file reveals directories.
- Certificate transparency reveals subdomains.
- Search results expose old pages.

These are useful recon signals, but they are not permission by themselves. They only tell you something exists. The program policy tells you whether you may test it.

This distinction matters most with:

- partner APIs
- fulfillment or logistics systems
- payment-adjacent systems
- identity providers
- support portals
- third-party SaaS integrations
- cloud storage buckets
- old subdomains
- documentation generated for internal or partner use

When an asset looks sensitive, slow down. Confirm scope before interacting with it beyond passive review.

Use A Stable Researcher Identity

Programs want to distinguish legitimate research from abuse traffic. Make that easy for them.

Use a consistent setup:

- one primary browser profile
- one testing account set
- one researcher email or approved alias
- one normal IP source when possible
- one device or stable workstation
- low manual traffic while validating behavior

Avoid behavior that looks like evasion:

- Tor exit nodes
- rotating proxies

- frequent VPN hopping
- cloud proxy pools
- scanner farms
- many regions or countries in a short time
- rapid ASN (Autonomous System Number) changes
- high-rate automated fuzzing without permission

A stable home connection is usually better than a noisy anonymity setup. A single dedicated VPS can be acceptable when you need stable logs or a fixed IP, but it should still be consistent, low-rate, and clearly tied to your research identity.

The operating principle:

`Do not disguise or distribute your testing infrastructure.`

Two Owned Accounts Are Enough For Most Authorization Tests

For access-control testing, use accounts you control. A clean setup usually looks like this:

```
Account A: primary test user
Account B: second owned test user
Browser profile A: logged in as Account A
Browser profile B: logged in as Account B
Burp project: captures both flows with clear labels
```

Use the accounts to compare boundaries:

- Can Account A read Account B data?
- Can Account A modify Account B settings?
- Can Account A reuse Account B object IDs?
- Can Account A use Account B tokens or links?
- Can Account A skip workflow steps intended for Account B?

Stop immediately if a test exposes real third-party data. You do not need to collect more. The report should explain the minimal proof and avoid storing unnecessary sensitive material.

Tangible Security Risk Is The Report Standard

A report needs more than "something changed" or "something looks wrong." It must show a security consequence.

Good impact examples:

- unauthorized account changes
- unauthorized data access
- privacy violation
- access-control bypass
- privilege escalation
- sensitive workflow abuse
- improper token or file access
- ability to modify another user's settings or state

Weak impact examples:

- cosmetic UI issue
- missing header with no exploit path
- version disclosure with no practical risk
- self-XSS with no realistic escalation
- clickjacking on a page with no sensitive action
- brute-force claim without allowed testing or evidence
- "could be vulnerable" with no reproduction

The reportability question:

`Can I show a real broken security boundary using minimal, authorized evidence?`

If the answer is no, keep researching. Do not force a weak report.

Duplicate Still Means The Method Can Be Valid

A duplicate report can still prove that your methodology works. If the triage team says the issue was previously reported, same endpoint, same impact, and same vulnerability class, that means your finding was real but not first.

Use duplicates as feedback:

- The bug class was valid.
- The impact model was valid.
- The evidence was probably understandable.
- The target surface may be too crowded.
- You need better speed, less saturated assets, or deeper workflow coverage.

Do not treat every duplicate as failure. Treat it as signal. It tells you that your process can reach real bugs, but your target selection or timing needs improvement.

Confidentiality Is Part Of The Work

Most bug bounty reports are confidential unless the program explicitly permits disclosure.

Do not publicly share:

● Safe public phrasing is general:

- report screenshots
- endpoint paths
- reproduction steps
- payloads

- private program details
- internal communications
- proof-of-concept videos
- target-specific writeups
- report IDs
- raw request or response data
- screenshots containing user, account, or program data

```
I worked on broken access-control testing in an authorized bug
bounty program.
```

Unsafe public phrasing names the target, endpoint, bug path, impact, or proof.

When in doubt, ask:

- Does this reveal the company?
- Does this reveal the endpoint?
- Does this reveal a payload or reproduction path?
- Does this reveal report status or internal communication?
- Could another person repeat the test against the real target?

If yes, do not publish it without written approval.

Evidence Hygiene

Evidence should prove the issue without over-collecting sensitive data.

Good evidence is:

- minimal
- reproducible
- clearly labeled

- redacted
- tied to accounts you control
- captured at low request volume
- enough to show the broken boundary

Bad evidence is:

- bulk data
- third-party personal data
- screenshots with unnecessary details
- long raw logs
- unredacted cookies or tokens
- unclear request/response dumps
- exploit chains outside scope

Use this evidence pattern:

```
1. Show normal authorized behavior for Account A.
2. Show normal authorized behavior for Account B.
3. Show the single boundary test.
4. Show the unexpected result.
5. Show why the result creates security impact.
6. Redact identifiers, tokens, cookies, and personal data.
```

When To Stop

Stopping is a skill. A good researcher knows when more testing increases risk without improving proof.

Stop when:

- you reach third-party data
- you can modify another user's state
- you confirm unauthorized access with minimal proof

- the next step would be destructive
- the next step would send email, SMS, shipment, payment, or notification
- the next step would create load or denial-of-service risk
- the next step would leave the program scope
- you are unsure whether the action is allowed

Your report can say:

```
I stopped testing at this point to avoid accessing additional
data or causing
state changes beyond the minimum proof.
```

That sentence shows maturity.

Target Selection Rules

Do not hunt only where everyone else hunts. Mature main apps and obvious API surfaces may have high payout ceilings, but they also have high duplicate risk.

When ranking assets, consider:

- in-scope status
- sensitivity of data or workflow
- amount of business logic
- authentication depth
- number of object identifiers
- file or document handling
- invitation or sharing flows
- account-linking flows
- write operations
- duplicate likelihood

- rate-limit and automation restrictions
- third-party ownership risk

Good targets often have:

- sensitive workflow state
- documents or attachments
- account preferences
- invitations
- user-to-user relationships
- order or subscription states
- profile and privacy settings
- role or membership boundaries

Low-value targets often have:

- static content only
- public marketing pages
- no authenticated workflow
- no state-changing action
- no sensitive data
- issues excluded by policy

Manual First, Automation Later

Automation is useful after you understand the workflow. It is dangerous when used to discover the workflow blindly.

Start manually:

1. Read the scope. 2. Create the scope card. 3. Walk the user journey. 4. Capture baseline requests. 5. Identify objects and state transitions. 6. Compare Account A and Account B. 7. Test one boundary at a time. 8. Record expected and actual behavior.

Only automate after you know:

- the endpoint is in scope
- the request rate is allowed
- the action is safe
- the test will not affect real users
- the result can be interpreted reliably

Manual testing gives you judgment. Automation gives you scale. Judgment must come first.

Operating Checklist

Use this checklist before every serious test session:

```
[ ] I confirmed the asset is in scope.
[ ] I read the program rules today, not last month.
[ ] I know which tests are prohibited.
[ ] I know the rate limits or automation limits.
[ ] I am using approved test accounts only.
[ ] I am using a stable researcher identity.
[ ] I am not using Tor, proxy rotation, or noisy
infrastructure.
[ ] I have a clean Burp project or logging method.
[ ] I know what sensitive data must be redacted.
[ ] I know the stop condition for this test.
[ ] I can explain the expected security boundary.
[ ] I can explain the tangible impact if the boundary fails.
```

If you cannot check every item, fix the setup before continuing.

Reportability Checklist

Before writing a report, verify:

```
[ ] The affected asset is in scope.
[ ] The behavior is reproducible.
[ ] The issue affects a real security boundary.
[ ] The impact is more than cosmetic or theoretical.
[ ] The proof uses accounts or data I am allowed to use.
[ ] The request volume was reasonable.
[ ] The report does not include unnecessary sensitive data.
[ ] Tokens, cookies, account identifiers, and personal data are
redacted.
[ ] The report explains what I did and where I stopped.
[ ] The recommended fix matches the root cause.
```

Chapter Takeaway

Operational bug bounty work is not about running the most tools. It is about making controlled decisions under authorization.

The rule is:

`Scope first. Stable identity. Minimal proof. Tangible impact. Confidential handling. Stop before harm.`

If you follow that rule, your work becomes safer, easier to reproduce, easier to report, and more likely to be taken seriously.