Vulnerabilities 101: How to Launch or Improve Your Vulnerability Research Game

Joshua Drake, Zimperium
Steve Christey Coley, MITRE
DEF CON 24
Aug 7, 2016

Introductions

- About Josh
  - 20-years of VR, Ran iDefense VCP
- About Steve
  - CVE co-founder, “Responsible Disclosure” (sorry), CVSS, CWE, ...
- Why we are doing this
  - Currently, there is *way* more insecure code out there than researchers. This isn't guaranteed in 10 years, though.
  - We need more people looking at code that’s deployed in the real world
- What we hope to accomplish by doing this
  - Encourage more people to get involved

Disclaimers

- This is our opinion only
- Based on our own career experiences
- Others have their own opinions
- YOU… proving the cliche… are a unique snowflake
- You'll find your own way, but hopefully we can help you find it faster
- No new 'sploits here

What is a Vulnerability?

- Too many definitions
  - Roughly: “a mistake in software’s design or implementation that allows an ‘attacker’ to conduct activities that (1) affect other users and (2) are not explicitly allowed or intended by the developer or sysadmin.”
- “What do you have?” vs. “What do you get?” - there must be a difference
- “How much help must the victim give you?” (user interaction)
  - Automatic
  - “Normal” usage (e.g., clicking on a link is *normal*)
  - Was victim stupid/clueless? (“just copy this javascript: url into browser”)
- “How much luck do you need?”
  - ASLR, unusual configs, narrow race windows

What is a Vulnerability? (2)

- Vulnerabilities != Exploits (awesome @SwiftOnSecurity tweet)
  - A Vulnerability resides in the software itself, doing nothing on its own
  - An Exploit is a set of steps (possibly manual, or in the form of a program) that interacts with the software in a way that has a non-zero chance of successfully taking advantage of a vulnerability"

What is Vulnerability Research?

- “The process of analyzing a product, protocol, or algorithm in order to find unintended behaviors that allow an attacker to gain additional access to functionality or data that has not been explicitly approved by the product’s administrator.”
- This process may take minutes, days, months, even years
- Some people use “Vulnerability Discovery” to distinguish finding individual bugs in specific software versus more systematic/academic work
- Solving puzzles within puzzles, where you don’t know what the puzzle is when you begin

Motivations: Why Do Vulnerability Research?

- Knowledge / Wisdom
- Altruism
- Self-Protection
  - Aka Secure your systems by hacking them
- Fame / Notoriety
- Money / Career
- Power
- Fun / Lulz
- Realistically, it's a mix per individual
  - Not every researcher will share your motivations
  - Vendors might have experience or only assume certain motivations

Potential Work and Employers

- You can be a finder, extender, builder, fixer, defender, cataloger, coordinator, communicator, malware analyzer, risk evaluator, trend-discoverer, ...
- Note: not all work gets public recognition
- Just for fun (“hobbyist”)
- Yourself! - Bug bounties, black/gray market sales, etc.
- Consulting firms that value research
- Security product companies (for marketing value)
- Software vendors (product security or response team)
- Government contractors (or, government directly)
- Academia (focus is “pure” research)
- CERTs (analyze and understand real-world attacks)

Skills for Success

- You can have some of these skills/traits and get by, but we're trying to talk about what (usually) seems to lead to success
- Code, protocols, file formats, or "how things work" under the hood
- Common attack patterns
- Logical flows (e.g. logic vulns, CSRF, authZ/authN, etc.)
- How to run analysis tools and evaluate their findings
- Clear communication; one (preferably many) of
  - Steps to reproduce, and/or functional PoC with well-labeled functions, comments, etc.
  - Using common vocabulary
  - Describing the issues - first to vendor, then to public
  - Describing why it's important
  - Understanding and respecting your audience(s)
  - Well-structured advisory
  - Note that poor English is NOT on this list; the above items are much more important
- Clear communication is probably one of the biggest contributors to career success, no matter your specialty (also, less drama due to misunderstandings)

Personality Traits for Success: "Should" Have

- Persistence
- Patience - especially when dealing with people
- Diligence
- Curiosity
- Critical thinking
- Willingness to learn
- Self-motivated
- Willing & able to work independently, in solitary fashion

Personality Traits for Success: "Nice" to Have

- Collaborative
- High concentration
- Addictive
- Willing to share findings or techniques
- Passionate
- Desire for constant improvement
- Sense of humor

Key Terms (Vocabulary)

- Many of these terms have multiple definitions or usage
- Attack Surface: the set of all inputs and code paths with which an attacker can interact
- Impact
  - RCE (remote code execution)
  - EoP (escalation of privilege)
- PoC (Proof of Concept)
  - Ambiguous term…
  - What concept are you proving?!
  - Be clear, it will ease your efforts.
- Vulnerability classes
  - Memory corruption, injection (SQLi, XSS, etc.), protocol/specification design, …
  - When the low-hanging fruit fails: “Business logic”
- Root cause analysis
  - Ex: XSS in error msg indicating system() or path trav
- Chain analysis
  - It’s root cause turtles all the way down!

The Firehose: Where to Learn?

- OWASP Top Ten
- SANS/CWE Top 25
- White papers
- Periodic electronic collections
- Videos
- Mailing lists
- Github repos
- Vendor's bug databases
- Vuln scanners
- Intentionally-vulnerable packages
- CTFs / wargames
- Follow individual researchers
- Vulnerability databases
- Conference talks
- Classes
- Books
- Yearly White Hat Security Top 10 attacks

Selecting What to Analyze for Security Problems (1)

- You can go deep or broad
  - Language, vuln class, exploit technique, detection technique, ...
- Anything you do that contributes to the body of knowledge is valuable
  - Even negative results are useful! (though difficult to admit to)
- Lots of "low-hanging fruit" out there
  - Older code is more likely buggy
  - Complex or overly complex systems are often ripe
  - Large attack surface creates many opportunities
- Software popularity matters
  - Little-used software has lower quality but also lower impact to general public
  - Popular software with extensive vulnerability history is often difficult
- Too buggy means lower rewards or less recognition
  - Sadly, they won't get better without liberal application of effort

Selecting What to Analyze for Security Problems (2)

- Brand-new or emerging technologies
  - Vendor rush-to-market usually means security is at best an afterthought
- Newly-discovered or emerging vulnerability/attack classes
  - Each new class should force a review of ALL products across the board
  - Or, refine a new attack/vuln with new variations, stronger impacts, etc.
- Previously-unanalyzed code
  - Highly likely to contain lots of low-hanging fruit
  - Some targets grow popular without getting proper review / fixes (Android anyone? IoT?)
  - Some targets have been around forever but only recently connected to networks (hello medical devices and automobiles!)
- If you have access to expensive or difficult-to-obtain products: do eeeeeet
- Follow what others are doing ("Pigpile" or "Bandwagon" Effect)
  - Could offend the original researcher(s)
  - Benefit from a base level of published research

Tools and Techniques

- Dynamic vs. Static analysis - to run or not to run?
  - Dynamic is analyzing a program by running it - e.g. fuzzing, debugging
  - Static is purely inspection of program code - e.g. auditing, SCA tools
  - Code coverage (how much), accuracy (false/true positives) are important!
  - Real power is achieved by combining: hybrid analysis FTW!
- Code auditing (binary/source)
  - Grep! Pedantic compiler settings! Automated taint checking!
- Design review
- Threat modeling (e.g., STRIDE)
- Automated tools
  - Fuzzers, static code analysis
  - Risk of false positives
  - Lack of root cause analysis

Relevant Standards

- Using standards can make it easier to communicate critical vulnerability information across broad groups of people, including consumers, vendors, and others
  - (but haters do exist, and haters gonna hate)
- CVE - Common Vulnerabilities and Exposures
  - Numeric identifiers for tracking vulnerabilities
- CWE - Common Weakness Enumeration
  - Hierarchy of developer “mistakes” that lead to vulns
- CAPEC - Common Attack Pattern Enumeration and Classification
  - Common traits of attack methodologies
- CVSS - numeric rating
  - Pros: widely adopted, focused on key characteristics, provides consistency
  - Cons: not as consistent as hoped, difficult to use in non-traditional contexts

Disclosure Models

- Reasons for (public) disclosure
  - To inform the parties responsible for fixing
  - To put pressure on unresponsive vendors / get them to care
  - To inform the masses that there's a problem that needs attention
- Models
  - Full
  - Partial
  - Coordinated (formerly "Responsible")
  - Non-disclosure
- Standards Documents
  - ISO standard 29147 (@k8em0, etc.) - focuses on what VENDORS should do
    - Now freely available!
  - IETF Draft circa 2002
  - RFPolicy 2.0

Considerations for Your Disclosure Policy

- Your own disclosure policy can help clarify expectations between you and vendors
- What if:
  - You can’t even find the right contact point?
  - 0-day exploitation is actively occurring?
  - Somebody else publicizes your vuln(s) first
  - The vendor doesn’t respond?
- What is the correct grace period?
  - Design flaws often take a LONG time to fix

Considerations for Your Disclosure Policy (2)

- Impact to consumers who want to fix immediately
- Impact to consumers who can't fix immediately
- Whether vendor appears to be acting in good faith
- Will your actions:
  - Make it harder for others to want to work with you in the future?
  - Make it more difficult for people to hire you?
- "Is it worth it to disclose at all?"
- Again, no one-size-fits-all. Moral compass, etc.

Advisory Structure and Contents

- Your advisory is likely to be read by various people with slightly different goals
- The better your vendor coordination, the better your advisory details will be (for customers, anyway)
- Easily identifiable advisory structure
  - Well-labeled sections, different bugs in different segments, etc.
- Background / Explanation of software
- Synopsis / Abstract (brief)
- Affected software / hardware
  - Vendor name, product name, vulnerable versions
    - Newest vulnerable version
    - Fixed / non-vulnerable versions
  - Can quickly become a complex situation
  - Best effort in most cases

Advisory Structure and Contents (2)

- Privileges & access required to launch an attack
- Impact of a successful attack
  - Privileges gained; unauthorized operations that can now be conducted; etc.
  - CVSS score, along with full vector
- Detailed Description
  - "Too Much vs. Not Enough"
  - To PoC or not PoC? That is the question.
  - Level of detail is one of those individual opinion things
    - However, there is a real risk to disseminating attacks
    - Use your moral compass so you can sleep well at night
  - Typical PoC details: 1 or more of: affected parameter/filename, source code extracts (from input to vulnerable code), sample attack string (typically harmless), pseudocode, functioning exploit code segment, fully executable exploit program
- Patch availability, mitigations, workarounds
- Key identifiers (CVE, vendor IDs, CERT IDs, etc.)
- Disclosure timeline / Key Dates (discovery, research, patch, disclosure, etc.)
- Credit to contributors
- References to related work

Advisory Formats - Pros and Cons

- Simpler is better
  - Most widespread distribution; no special readers or software required
- Think “plain text” or “markdown”
  - Easy to copy-and-paste key details (also, language translation)
- Never, EVER PDF
- Video is a mixed bag
  - You’re taking up people’s time and limiting when they can see it
  - Clear picture is essential
  - Show steps for reproducing
  - Keep it short and sweet
  - Accompany it with a text advisory

What to Expect from Vendors

- tl;dr – everything and nothing
- MANY scenarios; every disclosure is a unique snowflake
- Inability to find right contact (who might not exist)
- Unless they're very experienced, you're calling their baby ugly
- Lack of understanding of the issue
- Acknowledgement of receipt, followed by silence
- Corporate bureaucracy / politics prevent open comms
- Refusal to share patches with you to re-test
- Lack of credits
- Commitment to a fix, but with an unreasonable timeline
- Disagreement on severity of the issue
- Release of patch without mentioning a vulnerability at all

Where to Disclose Publicly

- Post to at least one source that is archived forever (or archived widely)
- Mailing lists: Bugtraq, Full-Disclosure, oss-security
- Your own blog or website
- Exploit-DB or other exploit sites
- Vulnerability databases
- No separate publication - rely on vendor credits or “hall of fame”

Common Mistakes to Avoid

- Interacting with a vendor in a way that seems like a threat or blackmail
- PoC or GTFO ("Go! The Fail's Overwhelming!")
  - Easy to declare a vulnerability exists, but harder to prove it
  - Corollary: if you can't exploit it, maybe somebody else can
- Trusting automated tool findings without verifying them
- Skipping root-cause analysis
  - Often leads you to more interesting findings
- Not verifying whether the issue was already discovered

Common Mistakes to Avoid (2)

- Treating multiple attacks, or attack chains, as if they were separate vulnerabilities, even when they originate from a single vulnerability
  - E.g. if a vulnerability allows you to gain admin privilege, and a legitimate admin is explicitly allowed to "modify configuration" or "disable the software," then these abilities are NOT new vulnerabilities
  - Decision point: "if an issue is fixed, are the other issues still a problem?"
- Suggesting workarounds such as "uninstall software" is just… bad
- Over-hyping the severity of your findings
- Copying one of your old advisories to make a new advisory, and forgetting to change all the data for the new vulnerability
- Relying too heavily on memes or cultural references
- Assuming developers are stupid and lazy
- Assuming customers can patch instantly

VR Growth and Development (A Perspective)

- Disclaimer! Everyone develops differently, this is just an approximation
- Not everybody wants to be, can be, or needs to be elite
- Malcolm Gladwell's Outliers says it takes about 10,000 hours of focused practice to become an expert
  - Varies based on aptitude and prior experience, e.g. developers
- These days, it can take 3 years or more before you build a reputation
- To progress further, you can:
  - Team up with somebody else
  - Find a mentor
    - Be polite and respectful of their time; accept that some will say "no"
- QUIZ: what happens when you're an elite researcher who targets software with low-hanging fruit? Ask @taviso ;-)

Stage 1: Newbie

- Easy-to-find vulns
- Easy-to-conduct, simplistic attacks
- One vuln class only
- Misses more important vulns
- Misses nearby issues
- Finds and discloses each bug, one at a time
- Limited to highly insecure, previously-unaudited software
- No “advisories” per se
- Sometimes wrong

Stage 2: Workhorse

- More comprehensive findings - multiple bugs per package
- Multiple types of well-understood vuln/attack classes
- Recognizes simplistic protection mechanisms e.g. blacklists
- Evolves a disclosure policy and approach to working with vendors (or not)
- Evolving, stable advisory format
- Learns new techniques from others and applies them to own work
- Ensures findings are new and references related work

Stage 3: Subject Matter Expert

- Significant experience in one or more vuln or attack classes
- Develops new enhancements for existing techniques
- Writes white papers / speaks at conferences
- Bypasses common protection mechanisms
- Performs more comprehensive root cause analysis
- Applies experience to previously-uninvestigated product classes
- Creates a noticeable body of work
- Extensive findings for any package audited
- Experience with multiple techniques & methodologies
- Able to find bugs in most packages
- Detailed, well-written advisories with all relevant information
- Rarely wrong

Stage 4: “Elite”

- Hates that term (probably)
- Finds new vuln classes, invents new attack classes, makes new tools
- Bypasses state-of-the-art protection mechanisms
- Anticipates industry-wide developments
- Is “elite” only for a particular specialty
  - NOBODY knows everything anymore
- Finds vulns in any software package, anywhere, anytime*
  - * as applied to their particular specialty
- Analyzes most popular, secure software
- Finds complex vulnerability chains

Feelz and Failz: Your “Objective,” Technical Research is a Lie

- Vulnerability research is a trying profession/hobby
  - FAILZ are inevitable
- You’re (probably) not a robot
  - FEELZ are inevitable
  - You're (probably) subject to trying to find rationales and logic to explain away your feelz
- HACK/LIFE BALANCE IS KEY; but your balance ! = others’ balance
- You don't have to be l33t to make a difference

FEELZ ARE OK

- It's normal to:
  - Get frustrated
  - Get “scooped” by bug collisions
  - Be defeated by a technical barrier you don't understand
  - “Waste” time on a promising theory that doesn’t work out
  - Not understand what someone else’s advisory says
  - Give up, temporarily or permanently, and look for something else
  - Believe that your only worth is in finding RCE in software from Top 10 vendors
  - Not be able to see yourself reaching the level of those you respect
  - Boldly declare you're awesome (Dunning-Kruger effect)
  - Think you don't know much when everyone else believes you do (also Dunning-Kruger effect)
  - Let your pride & ego get in the way of communication
  - Ghost the disclosure process when you realize you're wrong
  - Think you found something new that's actually old
  - Get criticism from researchers you respect
- Try to prevent your feelz from negatively affecting anyone… including yourself

FAILZ ARE OK

- Your research heroes and heroines, plumbers and rock stars, whoever they are, *probably*:
  - Failed before everybody cared about infosec
  - Failed privately
  - Operated in a world where the “rules” weren't yet defined… but today those rules aren't made explicit
  - Over-hyped some finding or another
  - Got smacked down by somebody who raised questions they couldn't answer
  - Recovered, and forgot how they messed up
  - Recovered, but won't tell you how they messed up (see: ego)
  - Might misrepresent accomplishments or how easy things were for them
- Failz are not fatal! (usually; we *are* in the age of IoT, unfortunately)

Conclusion

- May you fail fast, fail uniquely, and fail well!
- Everybody forges their own path, but others have made the journey before
- Good luck and have fun!

Josh: @jduck

Steve: @sushidude

References/Links: Research Process

- Presentation
  - Andrew M. Hay - "Bootstrapping A Security Research Project"
    - https://speakerdeck.com/andrewsmhay/source-boston-2016-bootstrapping-a-security-research-project
  - Larry Cashdollar - "How to find 1,352 WordPress XSS plugin vulnerabilities in 1 hour (not really)"
    - http://www.wallofsheep.com/blogs/news/tagged/defcon#larry
  - Nick Jones / MWR Labs, "Bug Hunting with Static Code Analysis"
    - https://labs.mwrinfosecurity.com/assets/BlogFiles/mwri-bug-hunting-with-static-code-analysis-bsides-2016.pdf
- Books
  - Dowd, McDonald, and Schuh: "The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities" (the code auditing bible!)
  - "Hacker's Handbook" series, e.g. Drake, Lanier, Mulliner, Fora, Ridley, Wicherski: "Android Hacker's Handbook"
- Documents
  - Phrack Magazine: http://www.phrack.org/
  - PoC||GTFO https://www.alchemistowl.org/pocorgtfo/
  - "Introduction to Vulnerability Theory" - https://cwe.mitre.org/documents/vulnerability_theory/intro.html

References/Links: Tools

- This is far from exhaustive; there are dozens of commercial and freeware software scanners
- Consider: $$$, false-positive rate, false-negative rate, explanations, ...
- Kali Linux - many different tools https://www.kali.org/
- Metasploit https://www.metasploit.com/
- Grep (yes, grep!)

References/Links: Intentionally Vulnerable Software

- OWASP WebGoat https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project
- NIST SAMATE test suites, e.g. Juliet and STONESOUP https://samate.nist.gov/SARD/testsuite.php
- CWE "Demonstrative Examples" for individual entries https://cwe.mitre.org
- Intentionally vulnerable distros, e.g. Damn Vulnerable Linux or https://www.vulnhub.com/

References/Links: Advisory & Disclosure Advice

- Kymberlee Price, “Writing Vulnerability Reports that Maximize Your Bounty Payouts”
  - https://youtu.be/zyp2DoBqaO0
- John Stauffacher, “Geekspeed’s Advice for Writing a Great Vulnerability Report”
  - https://blog.bugcrowd.com/advice-for-writing-a-great-vulnerability-report/
- OSVDB “Researcher Security Advisory Writing Guidelines”
  - https://blog.osvdb.org/2013/01/15/researcher-security-advisory-writing-guidelines
- CVRF (Common Vulnerability Reporting Framework)
  - http://www.icasi.org/cvrf/
- Christey advisory format suggestion (2003)
  - http://www.securityfocus.com/archive/1/344559

References/Links: Disclosure Processes

- http://howdoireportavuln.com/
- http://attrition.org/errata/legal_threats/
- ISO 29147 vulnerability disclosure standard http://www.iso.org/iso/catalogue_detail.htm?csnumber=45170
- Christey/Wysopal IETF draft https://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00
- RFPolicy 2.0 https://dl.packetstormsecurity.net/papers/general/rfpolicy-2.0.txt