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

Joshua “jduck” Drake, Zimperium
Steve Christey Coley, MITRE
DEF CON 24
Aug 7th, 2016
Introductions

• About Josh
  – 20-years of VR, Ran iDefense VCP
• About Steve
  – CVE, “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
  – We hope to 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?

• Vulnerabilities ≠ Exploits
  • 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

Matt Holt
@mhol6
Stop using "vulnerability" and "exploit" interchangeably. Taylor Swift @SwiftOnSecurity already went over this.

https://twitter.com/mholt6/status/529797274658807810

Steve
What is a Vulnerability?

• Too many definitions...
  – Roughly: “a mistake in a system’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
Vulnerability Properties

- “How much help must the victim give you?” (user interaction)
  - None - Automatic
  - Low - “Normal” usage (e.g., clicking on a link is *normal*)
  - High - e.g. “copy this javascript: url into browser”

- “How much luck do you need?”
  - ASLR, unusual configs, narrow race windows

- Vulnerabilities have MANY other properties… Some will become apparent in the rest of this talk.
What is Vulnerability Research?

The process of analyzing a product, protocol, or algorithm - or a set of related products - to find, understand, or exploit one or more vulnerabilities.

- This process may take minutes, days, months, even years
- Vulnerability Discovery may be used to emphasize finding individual bugs in specific software
- Solving puzzles within puzzles, where you don’t know what the puzzle is when you begin
Why DO Vulnerability Research?

• Realistically, motivations vary per individual

  – Not every researcher will share your motivations

  – Vendors might have experience or only assume certain motivations

(word clouds generated on http://www.wordclouds.com/)
Vulnerability Related Careers

The need to deal with vulnerabilities spawns a wide variety of career opportunities
- This includes many roles that aren’t considered “pure VR” too
- Which job is right for you depends on skills, interest level, and more.
- Here’s a few of the myriad potential tasks:

● Discover ‘em
● Analyze ‘em
● Improve ‘em
● Exploit ‘em
● Investigate ‘em
● Catalog ‘em
● Fix ‘em
● Communicate ‘em
● Coordinate ‘em
● Minimize ‘em
● Prioritize ‘em
● Document ‘em

...we’ll elaborate more as we continue...
Potential Employers

• No one says you have to get a job doing this stuff. You could do it just for fun (aka a “hobbyist”).

• If you choose to get a job, you could work for (no particular order):
  – 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)
  – Regular businesses (internal application security teams)
  – Government contractors (or government directly)
  – Academia (focus is “pure” research)
  – CERTs (analyze and understand real-world attacks)

• Note: not all work gets public recognition
“Should Have” Personality Traits

- Patience - especially when dealing with people
- Persistence & patience - there will be challenges
- Diligence - details and accuracy are important
- Curiosity - fuel for motivation to learn
“Nice to Have” Personality Traits

- We feel these traits increase the chances of long-term, repeated success.
Skills for Long-Term Success

Certain skills, usually acquired over time, seem to lead to long-term success.

• Common attack patterns and logical flows (e.g. logic vulns, CSRF, authZ/authN)
• How to run analysis tools and evaluate their findings
• Clear communication; one (preferably many) of
  – Understanding and respecting your audience(s)
  – Well-structured advisory
  – Describing the issues (for the vendor and/or public)
    • Using common vocabulary
    • Why it’s important
    • Steps to reproduce, functional PoC with well-labeled functions, comments, etc.
  – NOTE: perfect English is NOT on this list; the above items are much more important
• Knowledge of “how things work” under the hood: code, protocols, file formats, ...

Clear communication is probably one of the biggest contributors to career success, no matter your specialty (also, less drama due to misunderstandings).
Key Terms (Vocabulary)

• Attack Surface: the set of all inputs and code paths with which an attacker can interact
• Attack Vector: the route by which an attack is carried out (i.e. email, malicious link, etc.)
• Impact
  – RCE (remote code execution), EoP (escalation of privilege), DoS (denial of service)
• PoC (Proof of Concept)
  – What concept are you proving?! A vulnerability exists? A crash happens? Exploitable?
  – Clearly communicate what you’ve proven. It will ease your efforts.
• Vulnerability classes
  – Memory corruption, injection (SQLi, XSS, etc.), protocol/specification design, …
  – When the low-hanging fruit fails: “Business logic”
• Vulnerability chains
  – Chains Example: Integer Overflows leading Heap Overflows
• Root cause analysis
  – Ex. XSS in error msg indicating system() or path trav
  – Ex. In C/C++, this is often the first encountered instance of undefined behavior.
  – Deeper analysis - ask yourself ‘why?’

NOTE! Many terms have multiple definitions or context-specific usage too..

Steve + Josh
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
- Individual researchers
- Vulnerability databases
- Conference talks
- Classes
- Books
- Yearly White Hat Security Top 10 attacks
- Pwnie Awards Nominees
- Bug bounties

See bonus slides for specific references!

Josh
Selecting Your Target (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 Your Target (2)

• Brand-new or emerging technologies (think Mobile, IoT, etc)
  – 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
  – 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)
  – Sometimes offends the original researcher(s)
  – Benefit from a base level of published research
Techniques and Tools

• Design review
• Threat modeling (e.g., STRIDE)
• 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 and 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!
• 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

• Models
  – Full, Partial, Coordinated (formerly “Responsible”), Non-disclosure

• 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

• Standards Documents
  – ISO standard 29147 (@k8em0, etc.) - focuses on what VENDORS should do
    • Now freely available!
  – IETF Draft circa 2002
  – RFPolicy 2.0 circa 2000
Disclosure Policy Considerations (1)

Every disclosure is a unique snowflake. Your disclosure policy clarifies 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

Image: http://feelgrafx.com/group/snowflake.html
Disclosure Policy Considerations (2)

• Impact to consumers who…
  – want to fix immediately
  – can’t fix immediately

• Is the vendor 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; follow your moral compass
Advisory Structure and Contents (1)

Your advisory will be read by various people with different goals. Make it count. Better coordination means a better final advisory.

• 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 fixed and non-fixed vulnerable versions
    • Oldest vulnerable version
  – Best effort as it can quickly become a complex situation
• Vulnerability type(s)
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
  - Level of detail is one of those individual opinion things, but there is a real risk to disseminating attacks
  - Include PoC (details on next slide)
  - Use your moral compass so you can sleep well at night
- Patch availability, mitigations, workarounds
- Key identifiers (CVE, vendor IDs, CERT IDs, etc.)
- Disclosure timeline (discovery, notification, vendor reply, patch, disclosure, etc.)
- Credit to contributors
- References to related work
Proof-of-Concept Makes Your Case

This key detail often tips the scales, so to speak. Consider including 1 or more:

- Sequence of steps to reproduce, e.g. through the user interface
- Detailed code analysis
  - Affected parameter or field
  - Name of affected function
  - Names of all files/executables involved (eg. login.php & authlib.php)
  - Source/assembly code extracts (or Pseudocode)
    - How does attacker input get to the vulnerable code?
    - Code version
    - Source line numbers / addresses
    - Trim unnecessary code for readability
- Code that demonstrates impact
  - Harmless sample attack, e.g. calc.exe or alert(‘XSS’)
  - Functioning exploit code (snippet, full, or in between)
Advisory Formats - Pros and Cons

• Simpler is better
  – Most widespread distribution; no special viewer required
  – Complex formats mean fewer potential readers (due to higher possibility of vulnerabilities)
• Think “plain text”. “Markdown” is a decent compromise.
  – Easy to copy-and-paste key details (also, language translation)
• Please never, EVER PDF!
• If you’re going to make a video...
  – Respect people’s time!
  – Keep it short and sweet, and accompany it with a text advisory.
  – A clear picture is essential
  – Show reproduction steps
  – Give the viewer enough time to read and understand each step
What to Expect from Vendors

• There are many scenarios, each 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
  – Legal threats
  – Acknowledgement of receipt, followed by silence
  – Corporate bureaucracy or politics preventing openness
  – 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

Please post your advisory to at least one source that is archived forever (or archived widely)!

- Mailing lists: Bugtraq, Full-Disclosure, oss-security
- Exploit-DB or other exploit sites
- Vulnerability databases
- Your own blog or website

Then again, maybe you are okay with relying on vendor credits or “hall of fame” -- no separate publication needed.
Common Mistakes to Avoid (1)

• Testing live sites or networks without permission
• Interacting with a vendor in a way that seems like a threat or blackmail
• Failing to ensure a problem exists
  – E.g., trusting automated tool findings without verifying them
  – Easy to declare a vulnerability exists, but harder to prove it
  – Corollary: if you can’t exploit it, maybe somebody else can
• Not verifying whether the issue was already discovered
• Skipping root-cause analysis
  – Often leads you to more interesting findings
• Suggesting poor workarounds (e.g., “uninstall software”)
• Over-hyping the severity of your findings
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
  – Decision point: “if an issue is fixed, are the other issues still a problem?”
• Copying one of your old advisories to make a new advisory, and forgetting to change all the data for the new vulnerability
  – Start with an empty template!
• Relying too heavily on memes or cultural references
• Assuming developers are stupid and lazy
• Assuming customers can patch instantly
Vulnerability Research Growth Stages
(just our take)

Newbie
- Easy vulns
- Easy attacks
- Easy software
- Sometimes wrong
- No clean advisory format

Workhorse
- Mult vulns
- Mult vuln types
- Simple bypasses
- Evolving advisory format

Subject Matter Expert
- Enhances existing techniques
- Early adopter
- White papers / conf speaker
- Root cause analysis
- Clear body of work
- Rarely wrong

“Elite”
- New vuln classes
- New attacks
- New tools
- New bypasses
- Complex chains
- Most popular/heavily-audited software
- Highly specialized
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 pivoting to security

• It can take 3 years or more before you build a reputation

• To progress further, you can:
  – Team up with a peer
  – 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 ;-)
Feelz and Failz: Your “Objective,” Technical Research is a Lie

• Vulnerability research is a trying profession/hobby
• FAILZ are inevitable
• 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 each person has a different balance
• You don't have to be elite to make a difference
FEELZ ARE OK

• It's normal to:
  – Get frustrated
  – Give up (temporarily or permanently) and look for something else
  – “Waste” time on a promising theory that doesn’t work out
  – Let your pride & ego get in the way of communication or success
  – Believe you're an expert when you’re not
  – Feel you’re weak when others think you’re not
  – Be afraid your work isn’t worthwhile
  – Be unable to see yourself reaching the level of those you respect
  – Feel hurt or embarrassed by criticism from people you respect
• Try to prevent your feelz from negatively affecting anyone…
  – Including yourself!
• Try to temper your emotion and avoid premature celebration

Josh and Steve
• Your research heroes and heroines, plumbers and rock stars, whoever they are, *probably*:
  – Failed privately, or before everybody cared about infosec
  – Got “scooped” by bug collisions
  – Were defeated by a technical barrier they couldn’t overcome
  – Couldn’t understand somebody else’s findings
  – Operated in a world where the “rules” weren't yet defined… but today those rules aren't made explicit
  – Over-hyped some findings
  – Recovered, and forgot how they messed up
  – Recovered, but won't admit 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

Image: wocintechchat.com
Backup Slides / Details
Stage 1: Newbie
(Details)

- Easy-to-find vulnerabilities
- Easy-to-conduct, simplistic attacks
- One vulnerability type 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
(Details)

- 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

(Details)

- Recognizes multiple audiences
- Significant experience in one or more vuln or attack classes
- Develops new enhancements for existing techniques
- Writes white papers or 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 (able to be trusted without verification)
Stage 4: “Elite”  
(Details)  

• 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*  
• Analyzes most popular, secure software  
• Finds complex vulnerability chains  

* as applied to their particular specialty
Vulnerability Research Process (an attempt...)

**Discovery**
- Finding security bugs by applying various techniques
  - Auditing
  - Fuzzing
  - Manual testing
  - etc.

**Analysis**
- Understanding the bugs as much as possible
  - Reachability
  - Impact
  - Affected product(s) and version(s)
  - Minimize test cases
  - Exploitability

**Reporting**
- Communicating your research with vendors (and potentially the general public)
  - Drafting an advisory
  - Notifying affected parties
  - Coordinating with vendors
  - Once remediated, notifying a wider audience

**Remediation**
- Fixing the issues, typically handled by the receiving end (vendor).
  - Craft and deploy a fix
  - Notifying affected parties
  - Coordinating with researchers
  - Affected parties
    - Prioritize, test, and apply patches
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](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)”
  – Nick Jones / MWR Labs, “Bug Hunting with Static Code Analysis”

• Books
  – Dowd, McDonald, and Schuh: “The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities” (the code auditing bible!)

• Documents
  – PoC||GTFO [https://www.alchemistowl.org/pocorgtfo/](https://www.alchemistowl.org/pocorgtfo/)
• 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

• NIST SAMATE test suites, e.g. Juliet and STONESOUP

• 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”

• 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://howdoireportavuln.com/)
• Christey/Wysopal IETF draft [https://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00](https://tools.ietf.org/html/draft-christey-wysopal-vuln-disclosure-00)
• RFPolicy 2.0 [https://dl.packetstormsecurity.net/papers/general/rfpolicy-2.0.txt](https://dl.packetstormsecurity.net/papers/general/rfpolicy-2.0.txt)