PATHFINDER REPORT

Bug Bounties and the Path to Secure Software

Exploring the role of bug bounties and vulnerability disclosure to create consistency, resiliency and efficiency in software and development practices

JUNE 2017

COMMISSIONED BY:

hackerone

About this paper

A Pathfinder paper navigates decision-makers through the issues surrounding a specific technology or business case, explores the business value of adoption, and recommends the range of considerations and concrete next steps in the decision-making process.

About 451 Research

451 Research is a preeminent information technology research and advisory company. With a core focus on technology innovation and market disruption, we provide essential insight for leaders of the digital economy. More than 100 analysts and consultants deliver that insight via syndicated research, advisory services and live events to over 1,000 client organizations in North America, Europe and around the world. Founded in 2000 and headquartered in New York, 451 Research is a division of The 451 Group.

NEW YORK
1411 Broadway
New York, NY 10018
+1 212 505 3030

SAN FRANCISCO
140 Geary Street
San Francisco, CA 94108
+1 415 989 1555

LONDON
Paxton House
30, Artillery Lane
London, E1 7LS, UK
+44 207 426 1050

BOSTON
75-101 Federal Street
Boston, MA 02110
+1 617 598 7200

Introduction

Publishing software is a risky prospect. Competing priorities push well-established brands and start-ups alike to release more software on an increasingly accelerated schedule. When a website or application is released on the Internet, it is an unfortunate but foregone conclusion that attacks will immediately commence and continue indefinitely. Given the commonality of real-world security flaws and third-party attacks, any program that solicits users to report issues must be carefully crafted, with clear guidelines and expectations.

In many ways, creating a bug bounty is the smart and logical move. The CISO knows the web application is getting attacked anyway. Success in defending it is time-based: find and fix the holes before malicious parties discover them. The bug bounty is an opportunity to offer an incentive for reporting the flaws instead of exploiting them for profit. However, it would be a mistake for the unprepared organization to use a bounty impulsively or as a first step.

In the past, bounties offered hackers a win-win: permission to indulge otherwise harmless hacking and exploration as long as a few rules were followed and any bugs discovered were reported. The reporting process between bug hunter and corporation hasn't always been a smooth one, but things have changed. The bug bounty process became organized and standardized. Bug bounty platform providers (BBPPs) emerged. Bug bounties are becoming accepted as a normal part of the software development lifecycle (SDLC). BBPPs help guide companies toward more secure software and help ensure fair treatment for the researchers and hackers reporting vulnerabilities. Are bug bounties for everyone, though? We explore the answer to that question and many others.

ABSTRACT

This paper focuses on strategies built around harnessing the often impatient but generally benevolent abilities of the hacker mindset. The paper argues that the power of this mindset, especially in large numbers, can be used to discover and mitigate serious bugs and vulnerabilities in software before organized cybercrime and malicious individuals have an opportunity to exploit and profit from them. Bug bounties and vulnerability disclosure programs have demonstrated their contribution to improving software security and reliability, both prior to and during production use. In early 2016, for example, Facebook reported a 38% increase over 2014 in the discovery of high-impact bugs thanks to their program, while Google's Bughunter University noted a 50% increase in reward payouts in 2016 over 2015. Successful use cases exist for organizations of all sizes and in all industry verticals. For example, given the implications not only for public safety but for the safety of human life, trends suggest that security assessments might one day be required for 'smart' technologies that contribute to the emerging Internet of Things (IoT).

Not everyone is comfortable with the idea of opening up proprietary products and software for strangers to analyze, explore and critique. Getting used to the idea that researchers can be allies, not adversaries, takes time and results in a number of questions. It has been generally observed throughout the growth and domination of software-driven businesses that any publicly exposed assets (software, infrastructure, apps, devices) will inevitably be explored, tested and/or attacked. Accepting this to be true, running a bounty program becomes the difference between discovering what others know or could find out, and remaining in the dark. Put simply, the common refrain in the industry is: "Everyone gets a free penetration test – whether or not they get a copy of the report is up to them." Companies eventually come to the realization that the difference isn't in how or where a budget is spent to balance security with development cost; rather, it's how a company chooses to respond when vulnerabilities become apparent that makes all the difference, especially when dealing with researchers.

Introduction to bug bounties

The path to enlightenment – and better software quality.

It's inevitable that software will be developed with bugs: unintended or undesirable behavior resulting from software code. When software defects can be exploited to compromise security, they become vulnerabilities.[1]

Most software development organizations maintain their own processes for discovering and fixing software defects. Attackers, of course, have long made it their business to discover and exploit vulnerabilities. But when third-party researchers working to improve security with no intention of exploiting bugs do so, they are faced with a challenge the attacker never has to worry about: how best to work with the software provider to resolve the issue?

The road toward an answer to that question has not always been smooth. A software provider has little reason to trust an unknown researcher, and companies do not have a reliable and consistent means to validate a researcher's findings, or to differentiate a capable researcher from a spammer or adversary. Time is also of the essence, since the malicious may find and exploit an exposure before it can be resolved. This has sometimes led to public disclosure of a vulnerability, often with full disclosure of details, with or without coordination with the provider.[2] There is a history of animosity, frustration and legal action between companies and individuals over such an approach. Historically, companies have seen full disclosure as an unreasonable action, while individuals have accused companies (not without reason) of trying to keep issues quiet and trying to avoid the expense of addressing them.

WHAT IS A BUG BOUNTY, AND WHEN CAN IT HELP?

To balance business demands with the need to resolve an urgent priority, enlightened organizations have sought to develop more methodical approaches to working with independent researchers, including providing incentive in the form of a 'bounty' – compensation for providing information about an actual vulnerability.

The concept dates back to Netscape's original 'bugs bounty' initiative, begun in 1995, with major vendors introducing their own programs over the years since. Bug bounties cannot, however, be offered consistently and reliably in such as a way as to cultivate the participation of legitimate and qualified researchers without undertaking the processes and responsibilities of running a formalized program – and this is not a software maker's primary business. This has created an opportunity for third parties to manage bug bounty programs (BBPs) for software and technology makers, and has given rise to the market of commercial bug bounty program providers of today.

A few questions and considerations can help prioritize when a bug bounty might be necessary. The chief concern is the likelihood that a bug would result in an undesirable outcome. A breach is only one potential security-related outcome; others could be related to safety or proper operation. For example, cheating is a serious issue for online multiplayer gaming. While cheating isn't strictly a security-related issue, it could have a detrimental effect on subscriptions. If not fixed promptly, players could lose patience and leave the game permanently. How, then, do we determine the likelihood of an undesirable outcome? The most significant considerations may include:

1. VISIBILITY – Awareness is a big component in determining the likelihood of attacks against software. The more people aware of the software, the greater the likelihood of someone accidentally running into a critical bug or someone deciding to target the system for nefarious purposes. Websites, for example, represent a particularly exposed class of software, given their accessibility. Many in the security industry generally scoff at 'security through obscurity,' but we're not suggesting anyone try to hide their software. Rather, it's just important to be aware of how visible it is to the general public. Is anyone discussing the software in forums or on social media? More particularly, are they discussing how certain functions work, and potentially how they could be abused? This sort of intelligence simplifies prioritization.
2. CRITICALITY – The nature of software or the data handled by it is another key factor in determining the overall likelihood of attracting trouble. It is a common refrain for companies to claim to have "no payment data, no personally identifiable data – nothing an attacker would want" as a justification to avoid making security a priority. In reality, it is common for attackers to transfer money directly out of a company's bank account or to hold systems/networks for ransom by locking out administrators. Criticality also refers to the potential impact to the business if the service goes down or a data integrity issue occurs.

[1] Distinctions between bugs and vulnerabilities are further discussed in the Appendix.

[2] Differences in the ways vulnerabilities are disclosed are also discussed in the Appendix.

3. NOTORIETY – This factor is not like the others. While visibility and criticality generally have a constant value, notoriety can result in negative PR and equally negative public feedback. Managing notoriety is ultimately a PR responsibility, but cannot be successful without cooperation from other parts of the business as well.

Any company finding itself ranking high on any of these factors, or at a moderate level across all three, should consider a bug bounty. In many cases, even a modest-sized bug bounty can yield useful results. Some of the most egregious vulnerabilities can be the easiest mistakes to make and the easiest for researchers to find. This low-hanging fruit is also what attackers are most likely to look for.

WHAT IS A BUG BOUNTY PROGRAM?

A bug bounty program is a formalized set of processes designed to engage researchers to evaluate software and discover its defects. Researchers are expected to report bugs and issues that could affect the performance, security or behavior of an application or system in exchange for recognition, gifts or money. A program should clarify the conditions under which incentives will be provided and offer a consistent and reliable means for researchers to submit findings and receive compensation, while formalizing the processes of analysis, validation and bounty reward for the subject organization.

Traditional security assessments often assign only one or two consultants to any given effort, and it's unlikely that any single security consultant would have expertise across the entire spectrum of software types, languages and architectures. Therefore, a model that incorporates a pool of dozens or even hundreds of individuals with varied skills and backgrounds is more likely to produce the most comprehensive results.

As a tool, a BBP is most effective when an organization is already effective at resolving reported bugs and vulnerabilities, and has the resources to potentially sort through dozens, if not hundreds, of submissions. If the idea of hundreds of strangers suddenly submitting bug reports comes off as unsettling, that's probably a healthy response. While this paper explores the potential benefits of bug bounties, it also cautions against attempting to launch a bug bounty before ensuring the organization is prepared. Private bounties, however, can function as a compromise between the unpredictable volume associated with public bounties and the limited exposure of a traditional security assessment that only pits a single individual against an application.

A bug bounty platform provider (BBPP) helps with the design of BBPs and provides a software solution to manage submissions. Typical services include a SaaS-based platform for bug reporting, mediation services, a pre-existing talent pool and various support services to assist with the bug bounty management process. There are currently four BBPPs with a significant presence in the marketplace: HackerOne, Bugcrowd, Synack and Cobalt.

BBPPs offer expert guidance and implementation of a number of processes vital to the success of a BBP, among which vulnerability disclosure and handling are central. While any software maker may have its own processes for handling bugs and vulnerabilities discovered internally, the BBPP extends these processes in practical ways to outside investigators.

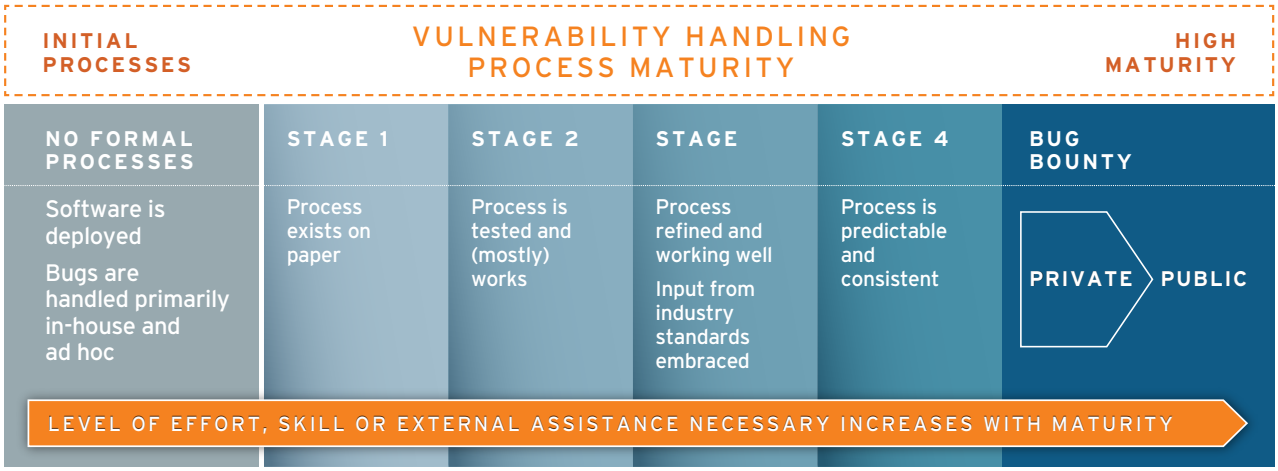
Making the most of a BBP

"Fortune favors the prepared mind." – Louis Pasteur

A bug bounty isn't a good choice for the unprepared. There's a path every organization should take to ensure all the right components are in place to handle the impact a bug bounty will have on development teams. The path to bug bounty readiness involves more than simply having the budget and human resources necessary – specific roles in the process need to be understood, the vulnerability handling process should be well-practiced, and timely, organized communication is critical. The process of maturing down this path looks like the following figure, from a high level.

Vulnerability handling isn't something a company is likely to nail the first time; like any other personal or corporate skill, it takes practice. Although it doesn't necessarily need to happen in four stages, the process may be envisioned as follows:

Figure 1: Vulnerability handling process maturity



Source: 451 Research

STAGE 1: Process exists on paper.

STAGE 2: Process is tested, and works (mostly).

STAGE 3: Process has been refined and is working well; input from standardized practices such as the ISO 29147 standard for vulnerability disclosure, and ISO 30111 for vulnerability handling processes (both discussed further in this paper), have helped.

STAGE 4: The organization can get through the vulnerability handling process in a predictable and consistent manner.

For software developers, it is useful to understand where bug bounty processes can have a positive impact on implementing security throughout the development lifecycle, in order to make the most of what a BBP can offer:

Everyone's first step: a vulnerability disclosure policy

The thousand-mile bug hunting journey begins with a single vulnerability disclosure policy.

Vulnerability disclosure is often defined by a vulnerability disclosure policy (VDP) – the language and rules that define how an organization chooses to receive and handle vulnerability reports. An effective VDP ensures that the general public knows how to disclose bugs and provides confidence that issues will be handled in an expedient and competent manner. An ISO standard, ISO/IEC 29147, was developed to address the vulnerability disclosure process, and is freely available on the ISO/IEC Information Technology Task Force website.

Regardless of opinion on the value of bug bounties, a VDP should be considered table stakes for any company with a public footprint. Disclosures may not be limited to software bugs; configuration errors or any other detectable aspect of exposure may be discovered, and should be handled according to policy whenever feasible. Making an effective communication channel available for good Samaritans is a wise move. A VDP could be as simple as an email address or contact form, although more detail would be ideal.

A vulnerability disclosure policy should include, at a minimum:

1. Contact information
2. A clear description of what types of issues can be reported
3. Rules for finding and reporting bugs
4. List of systems available to report bugs on
5. Communication expectations: when to expect to hear back after first contact
6. Rules of engagement: how much is okay, and how much is going too far (i.e., potentially breaking the law)
7. Guidance on how to test may also be provided, such as providing a detailed summary of the vulnerability, including the target, steps, tools and artifacts used during discovery. This information can help the vendor reproduce the vulnerability.

The US National Telecommunications and Information Administration (NTIA) has developed a template that organizations may consider when developing a vulnerability disclosure policy. This template includes areas such as:

- Brand promise ("The safety and security of our customers is important to us…")
- Initial program and scope: to outline which systems and capabilities are 'fair game' vs. 'off limits,' which may evolve as capabilities change
- "'We will not take legal action if…": clear, unambiguous statements to guide researchers' good-faith efforts
- Communication mechanisms and process
- Non-binding submission preferences and prioritizations
- Versioning of the policy

Again, the VDP can be as simple as a contact form or an email address to which issues can be reported. Some companies may never find the need for anything more complex. However, many will find the need for much more than a basic communication channel. When third parties make the time to notify a vendor of an issue, reciprocity is assumed.

The importance of a speedy response and initiating dialogue can't be overstated. In fact, from vulnerability disclosures to breaches, there aren't many security-related incidents that can't be improved by effective and transparent communication. For example, social media orchestration service Buffer's breach was arguably some of the best marketing the company had ever received. The company's expedient, transparent and customer-centric handling of the incident turned an unfortunate event into a marketing win, attracting new customers and drawing an outpouring of support and understanding from existing customers. Buffer's example isn't a case where better vulnerability disclosure could have helped. Rather, it's an example of how honest, transparent communication can make the best of the kinds of bad situations companies find themselves in when disclosure processes fail. And planning for failure is a good way to make the most of a bad situation.

More recent examples include Cloudflare's 'CloudBleed' bug, which was discovered by Google Project Zero researcher Tavis Ormandy, and GitLab's database incident. Cloudflare mitigated CloudBleed less than an ***hour*** after learning about the bug. GitLab's incident, like Buffer's, wasn't the result of a software bug. Rather, it was human error, but the company received positive coverage for its transparent handling of the issue, which even extended to a live YouTube feed and soliciting advice from commenters on Twitter and Reddit. This is an extreme case, but even a small fraction of this transparency and responsiveness can go a long way in winning over customers and the general public.

Less impressive was Samsung's response to ***40 zero-day*** vulnerabilities an Israeli researcher found in the company's Tizen operating system. The researcher, Amihai Neiderman, notified Samsung of the issues yet went months with nothing but an automated email response to indicate that the company had even received his reports. It wasn't until Neiderman was interviewed for an article by journalist Kim Zetter that a human at Samsung responded, though still with a boilerplate, "we take security very seriously" response. Incredibly, the boilerplate claimed that the company acts promptly upon receipt of any "credible potential vulnerability." Only after being contacted by a journalist for comment did Samsung spring to action and get in touch with the researcher. The Samsung case is a textbook example of what's broken in many large corporation disclosure processes:

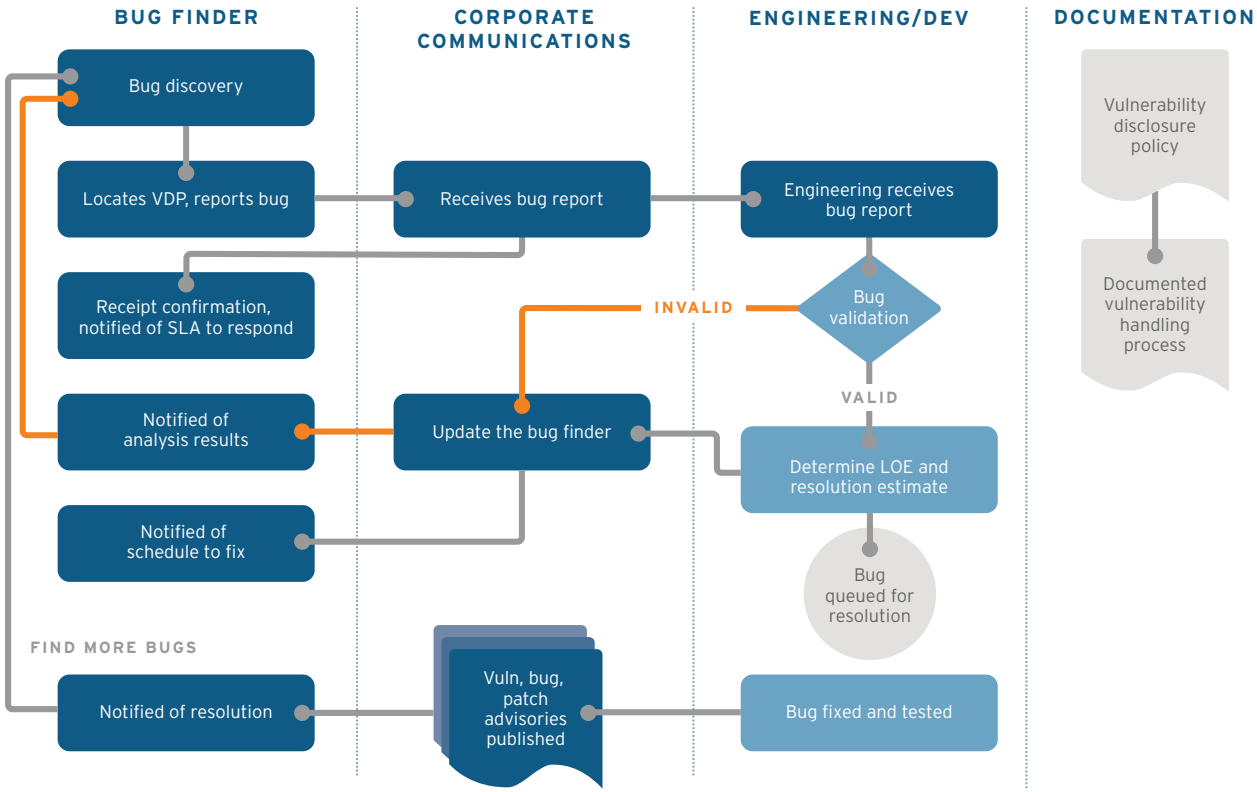
- There was ***no*** prompt response.
- Responses were wooden and patronizing, given the circumstances (claiming "we act promptly" after no contact for months).
- The response didn't match the gravity of the situation – Tizen is currently running on millions of devices, and the researcher didn't find one, but 40 zero-day vulnerabilities.

In addition to good communication, companies should also leverage the vulnerability policy to manage expectations. If communication delays will be inevitable at certain times (like holidays), make this clear, or automate responses during times when email may not be monitored. Finally, and most importantly, is the vulnerability handling process. This is the process behind the scenes that makes the vulnerability go away, and a smooth, well-practiced handling process is the key to preparing for a bug bounty.

VULNERABILITY HANDLING PROCESS (VHP)

The vulnerability handling process (VHP) is the process used by an organization to not only verify and resolve vulnerabilities once reported, but also to communicate with the reporting party. Thus, there are two main parts to a VHP – communication with the researcher and resolution of the bug. Below is a diagram of what both processes might look like, in a very general sense.

Figure 2: Vulnerability handling process



Source: 451 Research

Note that ISO standards 27147 and 30111 describe these processes in much greater detail, and separate the disclosure process (27147) from vulnerability handling (30111).

Providing an effective communications channel with the general public is essential, and is typically in any company's best interest. Consider separating these communications from standard public queries to ensure that responses aren't lost among lower-priority messages or forwarded to the wrong group. In many cases, a company's social media accounts are the easiest route to contacting the company. As such, the individuals monitoring these accounts should be prepared for recognizing and handling disclosures of bugs and other issues. There are countless examples of researchers trying and failing to report vulnerabilities due to improperly trained employees manning communication channels. The good Samaritan either gives up in frustration, or takes the one route guaranteed to get attention: full disclosure, often via social media or some mainstream media source. Communications channels for bug finders should therefore be clearly available and handled in such a way as to encourage cooperation from legitimate researchers – which, in turn, will help keep sensitive issues out of view of the malicious until resolved.

For many researchers, poor past experiences might shorten an individual's patience when reporting a vulnerability. As a result, an organization may only have days or hours before an impatient researcher goes public, so a quick response to disclosure could mean the difference between a PR nightmare and a copacetic, mutually beneficial relationship. To that point, **having a clear, common set of rules and expectations that is easy to locate is critical**. Setting expectations for a response within three business days, for example, could make all the difference in allowing an organization to have control over the timbre and outcome of the situation.

Bug bounty platform providers: key elements to consider

Bug bounties, including the use of bug bounty platform providers, have become commonly accepted as a valuable tool in development lifecycles. All new security markets have a period where they're viewed as a potential novelty or gimmick, until they either become standard tools in the toolbox, or are abandoned for the next big thing advancing down the industry roadmap. Not only have bug bounties joined the common lexicon, they've done so in a way that few other security markets have managed – with demonstrable evidence.

BBPPs reduce the burden of triage and response to reported bugs and give companies the necessary tools to automate many of the steps in vulnerability disclosure and handling, such as those specified by the ISO standards. Not all BBPPs are created equal, however, and companies must consider several key elements when choosing one (or several) to work with: disclosure assistance and community support, access to the pool of available talent, features native to the management platform, integration with existing workflows, and capabilities for automation and orchestration.

ELEMENT: DISCLOSURE ASSISTANCE AND COMMUNITY SUPPORT

The reality for many software providers is that security researchers are finding bugs every single day. Unless organizations have the resources and capability to manage the entire inbound process, interaction with these researchers can become overwhelming. The good news is that BBPPs provide a range of options for interaction with the researcher community. Simply providing a consistently monitored inbox for email submissions is a good basic start for many, and can be augmented with lightweight services that route emails through the BBPP's platform to help manage the receipt, verification and communication of inbound reports.

BBPPs should help manage relationships with researchers as well. It's in the nature of this community to test and probe software for weakness – but this mindset may be very different from the business world of the software provider. It can therefore be very helpful to tap experts in the community to help with communication and expectations. BBPPs that offer mediation or disclosure assistance provide valuable services that can help organizations handle these often unanticipated aspects of a BBP and avoid unnecessary pitfalls in cultivating what can be highly valuable relationships.

While evaluating a BBPP for vulnerability disclosure services, conformity with defined specifications should not be overlooked. The investment made in standards such as ISO 29147 and 30111 are specifically intended not only to give organizations the advantage of experience in best practices, but also to set reasonable and consistent expectations for researchers. BBPPs that embrace industry standards are well positioned to align with the software provider's reporting and processes – key factors that can alleviate many pain points, and should therefore figure prominently in discussions with candidate BBPPs.

Organizations considering a BBP should remember that they must navigate a two-sided marketplace: with the security,

development and remediation teams offering bug bounties on the one hand; and with the researchers hunting for bugs on the other. This highlights the value of BBPPs that take an interest in the broader internet community through effort like open source projects and consortia such as the Internet Bug Bounty, a program that encourages researchers to seek out and resolve weaknesses in the software and open source fundamental to the internet itself. BBPPs that offer support and services for these initiatives demonstrate commitment to the researcher community, and these are indicators of positive relationships with experienced researchers.

ELEMENT: TALENT

The ability to attract talent to a BBP is vital to its success. Even more important is the size, experience, quality and diversity of this network. Some companies may require a tester with experience in embedded system testing or experience in a specific programming language. The broader the experience of the available testers, the higher the likelihood that a company can find the right combination of skills to test their environment effectively.

Of course when opening a program to third parties, ability and experience levels are bound to vary widely – and when open to the public, aspiring participants may range from teenagers just starting out to security professionals with decades of experience. This also means that the most capable researchers are likely to be a minority of participants. While a few at the higher end have turned freelance into a full-time job, platform providers indicate that the majority of participants engage in research for bounties on a part-time basis. Bugcrowd indicates that about 70% of its researchers spend less than 10 hours per week on bounties, while 59% of the 617 researchers polled in HackerOne's 2016 Bug Bounty Hacker Report spend fewer than 20 hours per week.

One major difference among BBPPs is how they deal with these differences in capability and experience in fielding researchers. Synack and Cobalt, for example, are relatively exclusive and put hackers through interviews and skills assessments before allowing them to participate in the programs on their platforms. As a result, these companies' research teams are very small: Synack, which appears geared toward the traditional penetration test, indicates that its Red Team is about 105 strong, while Cobalt, which appears to be mostly focused on web-based applications, claims a Cobalt Core team of about 200 security researchers. This reduces the number of eyes looking at a company's code, but theoretically increases the likelihood that an individual report identifies a significant vulnerability. While Synack will only allow its vetted researchers to hunt for bounties, Cobalt allows its customers to open its programs up to all researchers, but recommends they stick to the company's core team.

At the other end of the spectrum are HackerOne and Bugcrowd, which have by far the largest and most diverse networks of researchers, covering everything from IoT to mobile applications, APIs and just about anything else running software in some form. These providers support public bug bounties in which anyone can participate. HackerOne claims the largest community, with over 100,000 security researchers and $17 million in bounties paid out to date. As noted previously, the most significant findings in any such broad community will likely be contributed by the more skilled and experienced minority. Among HackerOne's participants, more than 5,000 have so far received an award.

In higher tier commercial BBPP programs, a client may individually choose testers based on specific skills or other desired factors. Experience here is like a resume – is the tester really good at testing embedded hardware, or finding bugs in Java apps? What's the tester's rate of success? How happy have other clients been with the tester's work? Are a number of narrowly defined skill-sets preferable, or a wider variety?

HackerOne and Bugcrowd both support private bug bounty programs, in which only proven researchers are invited to participate. HackerOne, for example, allows companies to set a minimum 'reputation' requirement for researchers seeking to submit vulnerabilities. Reputation is gained or lost each time a report is resolved depending on whether or not the bug was exploitable and had not yet been discovered. A researcher's 'signal-to-noise ratio' (SNR) is represented by their average reputation per report. (SNR is discussed in more detail later under 'Metrics.')

ELEMENT: MANAGEMENT PLATFORM

Mozilla's Bugzilla has served many software projects well over the years. However, organizations without the resources to manage an in-house bug tracking system can take advantage of the SaaS-based platforms managed by BBPPs. The platform streamlines the bug reporting and management workflow for both bug finders and bounty creators. The platform also helps by enforcing standardized reporting, educating both sides and displaying useful metrics related to both the researcher and bounty owner.

One of the primary features of any BBPP's management platform is the means it provides for researchers to submit bug reports that allow the company to monitor the progress of each report and communicate with researchers, as described in the ISO standards. As noted previously, approaches may be as simple as monitoring an email inbox, but with potentially large numbers of researchers participating in the most wide-reaching programs, organizations must consider the impact on their programs of potentially large numbers of inbound reports, and the challenges of identifying truly valuable findings in that mass of information – the very problems that BBPPs exist to solve.

Triage thus becomes an important factor in considering a BBPP. In evaluating these services, organizations must weigh the effectiveness of triage as well as the flexibility of options provided. Bugcrowd, for example, offers its own triage for incoming submissions to ensure they are in scope and are not duplicates, and alerts companies when a submission requires attention. HackerOne offers triage services as part of its fully managed option, but allows customers the flexibility of using in-house triage or another vendor's offering. The platform also provides granular controls over who is authorized to view reports and what level of researcher, based on reputation, is allowed to submit bugs. Cobalt also offers triage that features what the company calls 'smart filtering,' which seeks to minimize the 'noise' among submitted reports and alert only on those that indicate potentially legitimate issues.

Organizations considering a BBP will also want to know what types of reporting and analytics options are available that will allow them to evaluate the progress and success of their programs as a whole, as well as the individual researchers participating. HackerOne's standard dashboard continuously monitors the customer's team to keep track of response times as well as the number of reports in triage, those where disclosure is pending, and those that have already been resolved. More advanced analytics are also available, enabling customers to submit queries to track metrics pertaining to the bug bounty program's ROI, while custom analytics and reporting can be purchased as part of the fully managed option. In addition, HackerOne offers a 'Success Index,' which compares the customer's security posture to other companies of a similar size based on a few key metrics. Bugcrowd also offers analytics and reporting on key metrics such as the most common bug types reported, ongoing totals and the average criticality of vulnerabilities discovered. Bugcrowd can also provide visibility into program activity for companies to see the most active researchers for a particular program and the rate of submission so that companies can establish stronger relationships with loyal researchers. Cobalt takes a similar approach by ranking its researchers and the quality of their reports for companies to identify specific researchers to work with if they choose. Cobalt monitors activity across bug bounty programs, while Synack's client portal also offers a variety of reporting options and patch verification.

There is one additional aspect of management that dovetails with the two-sided nature of a BBP. Organizations may want to assure that a BBPP provides everything they need to manage bug submission, validation and resolution. But the bounty aspect of a BBP certainly isn't overlooked by the researcher. Getting researchers paid, reliably and consistently, is therefore a key aspect that software providers should take seriously as well. After all, the responsible management of tangible assets are central to the concept of a BBP – and should be central to BBPPs, too.

Organizations will therefore want to consider the processes a BBPP has in place to support bounty payments when earned – processes that can further offload the burden of BBP management for the software provider. Guidance on pricing vulnerabilities may be offered. The eligibility of the researcher should be verified. Submission and management of proper tax forms is recommended. Integration with preferred payment systems should be noted. Popular platforms such as PayPal, Coinbase or Payoneer are often supported, and provide a layer of insulation between the bounty provider, researcher and financial services. Tools and processes that integrate directly with these platforms provide added value and further ease burdens for both software providers and researchers.

ELEMENT: WORKFLOW INTEGRATION

As illustrated previously, bug bounty programs can have a positive impact at multiple points in software development; indeed, with the growing number of BBPs, they become an increasingly visible aspect of the lifecycle. This means that, for a BBP to deliver maximum value, it should integrate as seamlessly as possible into a variety of workflow tools, including the customer's own workflow management systems. With the rise of DevOps tools and practices that aim to automate and accelerate software processes – and as infrastructure itself becomes increasingly programmable – interoperability with preferred toolsets becomes vital.

Figure 3: Workflow Integration for Bug Bounty Programs

Source: 451 Research

Bugcrowd, HackerOne and Synack all publish an API to allow customers to integrate existing workflows into their BBPs, in addition to specific integrations with third-party products. For the most part, companies use these APIs to integrate their bug bounty programs with ticketing and GRC software so that vulnerabilities can be easily tracked and managed from either the provider's platform or the third-party offering. Cobalt does not publicly publish its API, but it does integrate with GitHub and JIRA for the same reasons.

In addition to its open API, HackerOne has a variety of integrations across project management, bug tracking, development and security tools including Bugzilla, MantisBT, Slack, JIRA and ServiceNow, among many others, while integrations with Okta and OneLogin facilitate seamless BBP access management for client staff. HackerOne also enables customers to create custom workflows to track vulnerability reports so that critical vulnerabilities are tracked independently of less serious ones, in addition to more advanced or complex workflow options.

Bugcrowd has similar integrations with JIRA and HipChat. The company's JIRA integration allows customers to automatically generate JIRA tickets after bugs are validated and track them throughout the process. Synack also offers JIRA support; vulnerabilities resolved using JIRA are also reflected as resolved directly.

ELEMENT: AUTOMATION AND ORCHESTRATION

One of the biggest and most common chores in dealing with a sudden, large influx of bug reports is identifying submissions that end up being duplicates, false positives or out-of-scope. In fact, a considerable percentage of bug reports don't have to be triaged or prioritized at all – they just need to be categorized and responded to. In Yelp's recent analysis of the first 100 days of a public BBP on HackerOne, the company categorized 322 reports (57% of the total) as duplicates or 'not applicable.' That's 322 reports that would have to be manually inspected and categorized without any sort of automation. Furthermore, 525 out of a total of 564 were not actionable, meaning they didn't result in any fix or change. That's 93% of the total reports that someone would have had to categorize and file. The case for automating at least part of the workflow is practically a given.

All of the major BBPPs, including Bugcrowd, Cobalt, HackerOne and Synack, provide triage for their customers. In addition, many of the easier bugs to find will be discovered by multiple researchers, but bug bounties only reward the first to identify a vulnerability. To reduce the burden of identifying duplicate reports and eliminate the chance that a bounty is paid more than once for the same vulnerability, BBPPs also build duplicate detection into their platforms. According to publicly available information, HackerOne accomplishes this with pattern matching, while Cobalt identifies duplicates with natural language processing. Duplicate detection may be a less complex issue for Synack than for public bounty providers because there are reportedly only 105 highly qualified researchers submitting vulnerabilities to its customers. Synack does, however, automate many processes for its researchers, through what it classifies as a vulnerability intelligence platform. The technology is designed to streamline the Synack Red Team's research processes and accelerate vulnerability discovery.

ELEMENT: FLEXIBLE PROGRAMS

Flexibility in the type of programs available, as well as in the features inherent to a specific program, is key to casting the widest net possible to attract the most customers. The most apparent way in which platform providers cater to specific needs is in enabling customers to launch public or private bug bounty programs, which all of them do with the exception of Synack. Between private and public programs are those that limit participation to researchers who meet certain prerequisites, as with HackerOne's reputation system and Bugcrowd's similar feature.

Program flexibility goes far beyond simply determining researcher eligibility, however. HackerOne, for example, offers customers four levels of service: Free, Professional, Enterprise and Fully Managed, with Professional being the most popular according to the company's website. Bugcrowd offers customers the option to control costs by choosing between ongoing or project-based programs and capping costs so the program either ends when the deadline passes or a pre-defined reward total has been paid.

ELEMENT: METRICS

There are a number of metrics that are important in gauging the success of a bug bounty program. Examples of bug bounty metrics include mean time to first vulnerability discovered, mean response time, total reports submitted, percentage of duplicate reports and total payout. For example, HackerOne quotes that 77% of its programs identify a vulnerability within the first 24 hours. In the first 100 days of a BBP it ran with Uber, the company said that 2,030 reports were submitted and $345,120.48 was paid out to researchers. The program also exhibited a 1:6 signal-to-noise ratio, which means there were six 'noisy' reports submitted for every one with a legitimate vulnerability. All of this information together would indicate that Uber was relatively successful in its program.

Care must be taken, however, when interpreting performance measurements in isolation. One of the simplest metrics to calculate is the total number of reports submitted, which would seem to indicate the program identified a large number of bugs. However, without knowing the SNR (the proportion of reports that were legitimate vulnerabilities), the number of total reports submitted may mean little. Even knowing the SNR or being explicitly told the number of vulnerabilities discovered in a given time period can be misleading without further context. For example, public bug bounty programs are likely to have worse SNRs than private ones since less experienced and less skilled researchers are more likely to be involved in a public program than a private one, and will typically submit a greater proportion of noise. In its most recent State of the Bug Bounty report, Bugcrowd quotes an average SNR for all programs of 18%, but the SNR of private programs was 29% while the public SNR was 13%. HackerOne distinguishes 'clear' signal from 'nominal' signal: 'clear' signal means valid security bugs resolved by the vulnerability response team, while 'nominal' signal means reports marked as 'won't fix' or duplicates of resolved issues. While not contributing to clear signal, many such reports are technically accurate based on the best

information available to the researcher. Using these definitions, HackerOne's public programs reported 16% clear and 43% nominal signal in 2015, while its invitation-only programs reported 38% clear and 47% nominal signal.[3] Comparison of these figures to 2015 reports of the SNR of public programs of GitHub (4%), Facebook (5%) and Google (7%) highlights the advantages of BBPPs. It must be remembered, however, that public programs may attract insight not otherwise obtainable, as well as engage previously unknown but promising researchers. Additional metrics such as reputation scores that rate researchers based on the validity of submitted reports can help further increase SNR.

Severity of vulnerabilities discovered is another important metric that can enter into developing a successful BBP. A public program may be most likely to identify a large number of non-critical vulnerabilities, while a private program is more likely to discover fewer but more high-priority issues. These factors should enter into program choices, along with the benefits of outsourcing technology and process to a BBPP.

RUNNING A BUG BOUNTY: RULES AND SCOPING

"...the code is more what you'd call 'guidelines' than actual rules." – Captain Hector Barbossa

When inviting a group of individuals to test a system, clear rules of engagement are important. The context, rules, scope and any caveats should all be documented and made clear to researchers. A first-time bug bounty runner might imagine that finding the time and people to fix all the issues would be the biggest challenge – but often it is sorting through the noise generated by testers. These individuals might not understand where the target application 'begins and ends' or the necessary context to determine whether a finding is a false positive or not.

The scope of a BBP draws the boundaries of where and when testing is allowed. For example, on a given website, perhaps blog.vendor.com is out of scope, because it points to Wordpress, a blogging platform. While WordPress can be self-hosted, it's just as often managed by a service provider – a third party that will almost certainly be out of scope and probably won't appreciate being 'assessed' without warning. Setting the scope and finding the Goldilocks Zone for a project is a difficult task, and will likely need to be tweaked over time. One dilemma is how to handle bugs and vulnerabilities in third-party code, platforms and libraries. These vulnerabilities are valid and affect the application directly, but the vendor can at best mitigate issues when they don't own the code that needs to be fixed. So, does the bounty owner accept the bug submission and pass on the issue to the third party? Do they reject the submission and suggest the submitter notify the third party? Or does the owner accept and mitigate the issue, not bothering to notify the third party at all?

Then there's the issue of vulnerabilities that aren't code defects at all – they might be configuration errors or limitations inherited by the application's environment or runtime. These, too, should be reported, but may not be resolved in the same way or through the same workflow as code changes.

Make the scope of a BBP too broad, and the result is too much noise. Make the scope too narrow, and critical vulnerabilities can be missed. A narrow scope also tends to reduce incentive for hackers, since a large number of testers with too little attack surface end up stepping on each other's toes, resulting in increased duplicate and near-duplicate submissions. Finding this fulcrum point may be challenging, but will return dividends in the quality of submissions received.

HANDLING DISCLOSURE

When software and systems are publicly exposed, bugs and vulnerabilities will be discovered. Some will be discovered accidentally, while others will be found by employees performing due diligence or hackers that are just curious and want to understand how something works.

Ultimately, the entire idea behind creating a vulnerability disclosure policy is to ensure there's a clear process for communicating issues.

For companies with the necessary tools and resources, a BBP encourages and accelerates the process of discovery, using external resources (researchers and hackers) to increase scale. The challenges arise when there is an imbalance in expectations, and this issue has been the subject of debate for decades, commonly known as the 'disclosure debate.' This debate typically pits ***full disclosure*** against ***responsible disclosure,*** now more commonly called 'coordinated' disclosure. (See Appendix for more detailed definitions of disclosure.)

[3] HackerOne's public programs reported 19% clear and 51% nominal signal in 2016, while its invitation-only programs reported 46% clear and 42% nominal signal. For YTD 2017 the numbers are as follows: 23% clear, 47% nominal for public programs and 51% clear and 38% nominal for 2017.

Coordinated disclosure essentially argues that the safest thing is to give vendors as much time as they need to fix the issues. Full disclosure advocates argue that giving vendors an inch too often results in them taking a mile, and that full disclosure provides the incentive necessary to get the bug fixed. Fans of coordinated disclosure dismiss full disclosure as too dangerous, creating an asymmetric situation that favors the attacker. The rub is that it will generally take the vendor much longer to create and ship a fix than it takes the attacker to write an exploit.

One caveat here is that many organizations practice a hybrid approach of sorts – vendors are given a certain amount of time to address the issue before full details are publicly disclosed. Google Project Zero gives 90 days, and Trend Micro's Zero Day Initiative gives a generous 120 days, while Carnegie Mellon's CERT gives vendors only 45 days before making the details public.

BBPPs can help increase the transparency of disclosure for organizations that take advantage of such capabilities. HackerOne, for example, supports disclosure for its clients, while its Hacktivity feature provides a centralized showcase of disclosed vulnerabilities, researchers, programs and bounty awards for vulnerabilities that have been closed by a public program. Community members can engage with this activity via 'upvotes' that are balanced in rankings against a factor that considers the time since they were last updated.

To summarize, in many ways, managing disclosure policies and bounty programs is a bit like being handcuffed to the tail of a tiger. Despite our best efforts, it can always turn around and bite if it wants to. People, as a general rule, are a bit more reasonable and predictable than tigers. Monetary rewards and public recognition are appreciated and can go a long way, but most just want to see issues get fixed. In the end, good communication, mutual respect and timely responses should keep the tigers at bay.

Maintenance and measuring success

Before bug bounty, chop wood, carry water. After bug bounty, chop wood, carry water. – Ancient Developer Proverb

The goal of a BBP isn't a 'finish line' – it is a reliable, repeatable process. While a bug bounty could be used at a specific point in the overall lifecycle of a product, it is most valuable when used as a permanent, perpetual addition to the software development lifecycle (SDLC). Individual testers get to know certain companies, their code, their preferences and their needs. Developers learn how to leverage bug bounties more efficiently over time, learning from mistakes and catching easier issues themselves.

In the long run, how do we measure success? We could measure it by monitoring the severity and/or quantity of bugs slain and vulnerabilities fixed. Or we could measure success by the amount of bounties paid out. The problem with both of these metrics is that they don't indicate whether any lasting improvements have been made. It makes more sense, then, to measure success by the number of bugs ***that no longer get to the bounty stage in the first place***. An improvement in this metric likely indicates an improvement in the quality of software overall. Fewer bugs to fix means less risk and a reduced cost of doing business.

So how do we measure an improvement in software quality? Here are a few ideas for metrics that could help, keeping in mind that it will take a few development cycles before a pattern will begin to form.

NUMBER OF ISSUES PER 1000 LINES OF CODE (LOC)

- Steve McConnell, author of ***Code Complete*** and other best-selling programming tomes, estimates the industry average at 15-50 errors per 1000 LOC
- Also consider that it is a common belief that the frequency of defects increases with the overall LOC. McConnell also notes that it actually takes approximately 12.6 times more effort to build a project 10 times larger. No economies of scale here.
- Logic suggests that the number of bugs and the effort necessary to locate and fix them probably follows a similar trajectory.

NUMBER OF CRITICAL FLAWS PER DEVELOPMENT CYCLE

It isn't just the raw number of bugs that's important – look at the list of bugs that have been submitted. How many of those are truly show-stopping items? Issues that would directly result in breakage, data loss or compromise? SQL and command injection vs. reflected XSS bugs? Seeing less of a certain vulnerability type over time is a good metric to track since it shows your program is working and software is becoming more secure. Recording the ratio of high-severity vulnerabilities per development cycle and tracking that over time will also give an indication of how your secure SDLC efforts are paying off.

TIME TO RESOLVE

- The level of effort and linear time necessary to get through the handling process.
- Keep in mind that some bugs take longer to fix than others. For example, according to data from a small study done by The Denim Group, stored XSS took 9.6 minutes to fix on average, while SQL injection took an order of magnitude longer, at 97.5 minutes.
- Also keep in mind that the code fix is the least of the worries. Prioritizing the fix, reviewing it, testing it and queuing it up for release add considerable overhead to getting through the handling process.
- Look into automation and other improvements that can minimize the linear time and labor necessary to get through this process. Increased handling efficiency enables more bug fixes.

Additional Considerations

Beyond the webapp

What else can/should we include in a bounty? We've discussed how the ubiquitous webapp dominates current bug bounties, but software is invading more and more devices, use cases and environments. AT&T's bounty is, aside from a few exceptions, wide open. The bounty covers nearly everything publicly accessible that AT&T owns or sells: DVRs, smartphones, websites, mobile apps – all are valid targets for the bug bounty. Mobile apps and web services are also popular targets, though closely related to their webapp brethren.

IoT devices and consumer products seem to have a high tendency for critical defects. Consumer products, with their lower price points and margins, tend to skip even the most basic due diligence before hitting shelves. This often results in flaws so jaw-droppingly basic (like the recent CGI-BIN command injection issue with Netgear routers) that it seems only a matter of time before attackers build another Mirai, but mostly out of consumer-class devices. A number of organizations are putting together best practices, standards and even challenges with prizes/bounties. Microsoft's **Project Sopris** and the FCC's **Internet of Things Security Challenge** are two examples of this trend.

While the proportion of testers proficient in hardware security assessment is much lower than the software-savvy equivalent, some companies, like Intel, have increased the bounties accordingly. On average, hardware-related bounties, especially for more critical bugs like those that result in remote code execution, are around four times that of their software equivalents. Of course it's all software, but finding firmware bugs proves to be considerably different than hunting for vulnerabilities in web applications.

Third-party and supply-chain software present a dilemma. While most licensing agreements place restrictions on the buyer's ability to properly assess a COTS product, the amount of third-party code running in the typical corporate network far outstrips the amount of code written in-house. This concerns more than just switches, routers, servers and appliances – the applications we consider to be 'homegrown' often comprise more third-party libraries and components than code written in-house.

WHEN TO TEST?

Ideally, we'd all like to find and fix serious flaws before an application goes to production, but that can't always be the case. Regardless, testing against pre-production code or production code in a testing environment allows testers to find more bugs, since they don't have to be concerned with impacting production users.

PUBLIC VS. PRIVATE TESTING

There are significant pros and cons with both public and private testing. With private testing, a company can have the luxury of finding and fixing the most serious flaws before production release, and with no one the wiser. A public bounty, however, can be very positive marketing/PR. The aforementioned AT&T bounty where the scope includes nearly all aspects of the business is an impressive and bold undertaking. It conveys a certain confidence and determination to improve quality and security that will likely attract customers.

Another consideration is the quality and relevance of submissions. Revisiting the example mentioned earlier, Yelp ran a private bounty for two years prior to the previously mentioned public bounty. In the private program, 57% of submissions were found to be relevant, and 31% were actionable. That's over three times more actionable reports, with 30% fewer reports overall. To be fair, the private program researchers did get first crack at the code base, so it's likely that two years of the private program vacuumed up all the low-hanging fruit. From that perspective, it's actually remarkable that the public bounty still managed to find 39 more actionable vulnerabilities in only 100 days, considering the private bounty's researchers had two years to comb through Yelp's software immediately prior.

In conclusion, Yelp suggests that all companies should start out with a private program, due to the ability to control all constraints. In a private program, there are opportunities to choose testers, limit the number of them, and work on improving vulnerability handling capabilities in private.

Conclusions

Compiling all this advice into a roadmap would look something like the following:

1. Create a vulnerability disclosure policy (VDP) and make it easy to find.
2. Ensure that staff managing corporate communications understand how to recognize and handle a disclosure.
3. Document and practice vulnerability handling.
4. Select your BBPP partner according to your needs or set up your own bounty program scope and processes.
5. Start with a conservative private bounty program until comfortable handling the size of a public program.
6. Launch public bounty.
7. Refine and expand bug bounty as needed.

And some closing advice to remember:

- A bug bounty is another tool for the toolbox. It isn't intended to be the only method for discovering and resolving vulnerabilities. Continue using vulnerability scanners, static analysis scanners and penetration tests in conjunction with a bug bounty.
- The vulnerability disclosure policy is table stakes – everyone should have one.
- Track success with the suggested metrics, or create custom performance indicators.
- Fast and friendly communications help keep customers happy, researchers happy and the PR team happy.
- The goal isn't necessarily zero bugs, but quick, consistent fixes and fewer critical bugs.

In evaluating each of these processes, organizations are advised to assess the level of effort, the coordination of activities and processes on multiple fronts (external coordination with third-party researchers, as well as internal coordination of bug and vulnerability resolution), and the need to responsibly handle the tangible assets involved in bounty reward determination and payment. Bounty programs do not necessarily have to grow for these processes to become cumbersome very quickly. Remember, these are not processes that have been central to the business prior to launching a bounty program, but they are vital to the program's success. For all these reasons, organizations owe it to themselves to evaluate the capabilities of BBPPs that can ease these responsibilities – or outsource them entirely – and provide the tangible benefits of BBPs that organizations value.

Appendix: Terms, definitions and concepts

BUGS OR VULNERABILITIES - WHAT'S THE DIFFERENCE?

The terms ***bug*** and ***vulnerability*** may be used interchangeably, but there is a difference. A software bug is unintended or undesirable behavior resulting from software code. Vulnerabilities ***can be*** (more on that in a bit) a subset of software bugs that result in a loss of confidentiality, integrity or availability related to the intended operation of the software. For example, unacceptably slow performance in a software application would be classified as a software bug, but not as a vulnerability. Conversely, however, not all vulnerabilities are software bugs.

For example, a software application could be intentionally designed without authentication/authorization controls because it is designed to be used in a secure and trusted environment. The lack of these controls ***could*** result in a vulnerability if the application were deployed into an open environment. This vulnerability would therefore not be the result of a software bug, since the software is functioning as intended. Configuration flaws, implementation errors and misuse (RTFM-class errors, we could say) result in vulnerabilities that increase risk just as seriously as input validation failures and privilege escalation flaws.

UNILATERAL AND FULL DISCLOSURE

Unilateral disclosure occurs when researchers independently make the details of a bug or vulnerability public. This sometimes leads to confusion with 'public disclosure,' but this is an inaccuracy when research and disclosure coordinated with software providers as well as that pursued independently by researchers alone can both be made public. When software providers participate in a bug bounty, for example, the provider and researcher may decide mutually to share details with the industry. Some detail may be redacted in such cases, but the decision is cooperative. Unilateral disclosure, on the other hand, is only as restrained as the researcher. If the detail is complete, the approach is considered 'full' disclosure.

COORDINATED DISCLOSURE

Coordinated disclosure is an approach to reporting vulnerabilities designed to give vendors the benefit of doubt, and an opportunity to resolve issues privately. Sometimes organizations push to keep knowledge of the issue private indefinitely, and in other cases (more common these days), details are made public after the fix has been released or put into place. The general consensus, however, is that without the threat of eventual public disclosure, many bugs would never get resolved. As a result, we often see what could be considered a 'hybrid' approach, combining full and coordinated disclosure. This hybrid approach is basically coordinated disclosure with a time limit. At the end of the time limit, full disclosure occurs. Google's Project Zero closely follows this model, for example, with a strict 90-day limit. Bug bounty program providers may be able to support variations on coordinated disclosure or a 'hybrid' approach, as their clients may require.

RESPONSIBLE DISCLOSURE

Although often seen as synonymous with coordinated disclosure, this term has long been considered unfair and 'loaded' by researchers. As such, it is widely considered to be retired in favor of its replacement, coordinated disclosure. The implication behind this term is that other forms of disclosure are irresponsible by nature, painting a broad and negative image of hackers, bug hunters and security researchers. Full control over public access to information about the bug or vulnerability has often been abused, leading to companies taking months or years to fix issues – sometimes never fixing them. In response, researchers rejected this term, favoring instead a term that emphasizes the collaborative process of getting bugs found, reported and fixed.

NON-DISCLOSURE

There are three reasons for the hacker to avoid any sort of public disclosure: they find a vulnerability but don't know how to report it to the vendor and choose not to disclose; they want to save a vulnerability for profit or some nefarious future use; or because a company prefers to resolve vulnerabilities in private – a fairly common scenario with private bounty programs. Non-disclosure is also the option favored by cybercriminals, black hat hackers, state-sponsored offensive teams and commercial exploit vendors. Non-disclosure is nothing more than the decision to keep a vulnerability secret, allowing the vulnerability to retain both monetary and offensive value. Non-disclosure tends to be a gamble, however. As time passes, the likelihood increases that others will discover the 'secret' vulnerability independently. If this happens, it must be used immediately since it will rapidly lose value as enterprises scramble to apply patches.

PARTIAL DISCLOSURE

Partial disclosure is either where only certain parties are privy to the fact that a vulnerability exists, or where only basic details are shared publicly, preventing potentially malicious parties from crafting a 'zero day' exploit from the published details, or perhaps only slowing them down long enough for patches to be created and rolled out. Sometimes full details are made available later; sometimes they never are. This may also occur with bounty programs.

VULNERABILITY SCANS

Vulnerability scans are performed by software tools that aim to automate the search for both known and unknown vulnerabilities. The first type of scanning tool, a network vulnerability scanner, largely searches for known issues in COTS (commercial off-the-shelf) products. These products are generally given a range of IP addresses to scan, and begin with a discovery scan to identify active hosts, and a port scan to identify active services. Once this 'map' of available hosts and services has been generated, a scanner will attempt to identify services to determine if they may be vulnerable to any ***known*** flaws.

Web application (webapp) scanners, on the other hand, search for both known and unknown issues in websites, web-based applications, and web or API services. These tools typically take a website URL as input and start out by crawling the website or web application to get a 'map' of resources to test against.

PENETRATION TESTS

While the technical definitions of bug bounty programs and penetration tests (pentests) set the two apart, both can vary so widely that they could be entirely complementary, mostly redundant or anywhere in between. ***Typically***, however, pentests differ in some important ways. While bug bounties usually have clear scope as to the breadth and depth a researcher is allowed to go, the entire point of a pentest is to explore exactly how 'deep' an attacker ***could*** get into a company. In other words, a pentest typically targets an organization, not just an application. In another example, whereas a bug bounty is typically focused on discovering flaws in 'homegrown' code and applications, a pentest aims to find any and every way an attacker could do damage or cause losses to an organization, regardless whether through proprietary applications or COTS applications and devices.

Due to the broad nature of a pentest (typically: some can be quite restrictive in scope), there's often not enough time to thoroughly explore the flaws in any one piece of software – especially proprietary apps that the consultant has never encountered before. In some cases, a more in-depth assessment of a proprietary application is added onto a pentest, as an additional service with dedicated time and resources. This specialized assessment can cost as much as (if not more than) the pentest, however – both of which typically far exceed the cost of running a bug bounty. This is one example of when a bug bounty is traditionally very complementary with a pentest. In a pentest, the client pays for the consultant's time, regardless of the results (although there are a few firms that charge based on results), whereas in a bug bounty, only valid bug reports earn a payment. Additionally, a pentest is typically performed by a single individual, often assigned based on consultant availability rather than specialization (the pentester with the best webapp testing skills might not have been available that week, for example). This is in stark contrast to a bug bounty that may go on for weeks or months – or in many cases, never ends. Dozens or hundreds of testers are likely to discover a broader range of issues than a single pentester with only a few hours to spend on an assessment.

Ultimately, one is not intended as a replacement or substitute for the other. A penetration test aims to provide a business with a holistic understanding of the methods and vulnerabilities that an attacker could use against the business. To achieve this, a pentester can attack from many vectors: from the Internet or from a cubicle inside headquarters (simulating an insider threat); pivoting from one asset to another, if possible; or even using social engineering techniques against employees to gain access. Most bug bounties, on the other hand, focus on attacks that come from the outside, and target a single layer – going deeper is typically against the rules.

Another selling point of the penetration test is that an experienced pentester won't just identify flaws, but can help a business understand how they fit into the larger picture – important insight when prioritizing the time and resources to address the issues. In addition, a pentest aims to only report true positives, whereas filtering out false positives and duplicate issues is a time-consuming task when dealing with the results of a vulnerability scan or bug bounty submissions.

	Bug Bounty	Vulnerability Scan	Penetration Test
Depth of Assessment	High	Depends on tool's capabilities and operator	Varies, time-restricted
Breadth of Assessment	High	Depends on tool's capabilities	Time-restricted
Asset Scope	Public assets	Public and private assets	Public and private
Experience of Testers	Broad range	Automated tool	Professional/experienced
Cost	Very flexible; only pay for results	Cost of tool: $2k - $100k+	$200-$500/hr or more; pay regardless of results
Logistics	Anytime, anywhere	Depends on tool's capabilities	Client typically pays travel expenses for on-site work