Orange Is The New Purple

By April C. Wright
For BlackHat USA 2017

Abstract:

This paper defines and traces the roots of a current gap between development teams and security teams, discusses ways to close the gap, provides tips for driving change and thoughts on maturity of this process.

1. Introduction

There is a gap in the mindsets of software builders and security teams.

Why does it feel like a struggle when security interacts with development teams? It is because we don't speak the same language. We don't share the same goals.

"The geography we have created is all about speed, convenience, and scale; Security is an afterthought."

— General Michael Hayden, retired head of CIA, NSA

As Security, we need to realize that this is not the Builder's fault. Some Builders are self-taught (including myself), however at many of the best schools in the US, Security is overlooked as part of Computer Science (CS) or Software Engineering (SE) curriculum. Cybersecurity is generally offered as a completely siloed degree, separate from SE or CS. SE classes pertaining to information security are usually separate classes, mostly offered as electives.

It doesn't seem surprising, then, that asking a Software Builder, "Do you avoid the OWASP Top 10?" can result in a blank stare or the dreaded question of "No, what's that?"

If you've ever worked as a Software Builder (as I have) or with them (as I do today), you probably know they are under very tight deadlines. They are sometimes given code to create that is out of context from the larger application. Time to release, efficiency, and

accuracy are their metrics, and bug counts may not have security prioritized over other priority of bugs.

How do we break down the barriers between Security Defenders and Software Builders? How do we become the same team?

Change starts with us. It is our responsibility and in our best interest as security practitioners to encourage and educate developers to work with us, not against us.

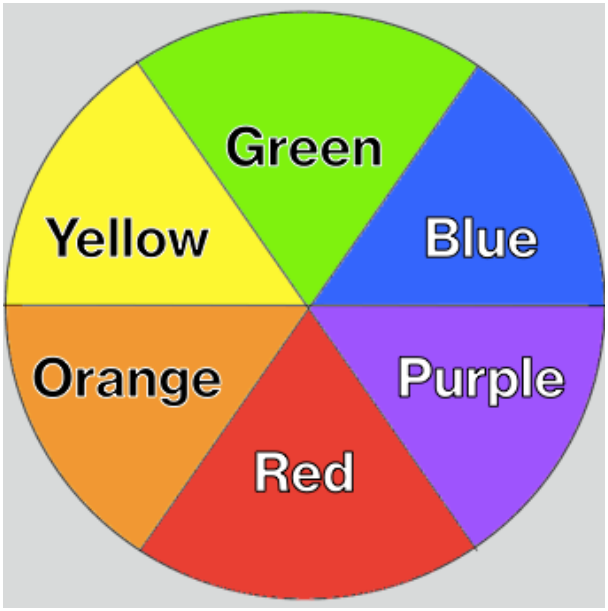
Why is this important? Because more secure software begins when we start bridging the gap between development and security.

2. Terminology

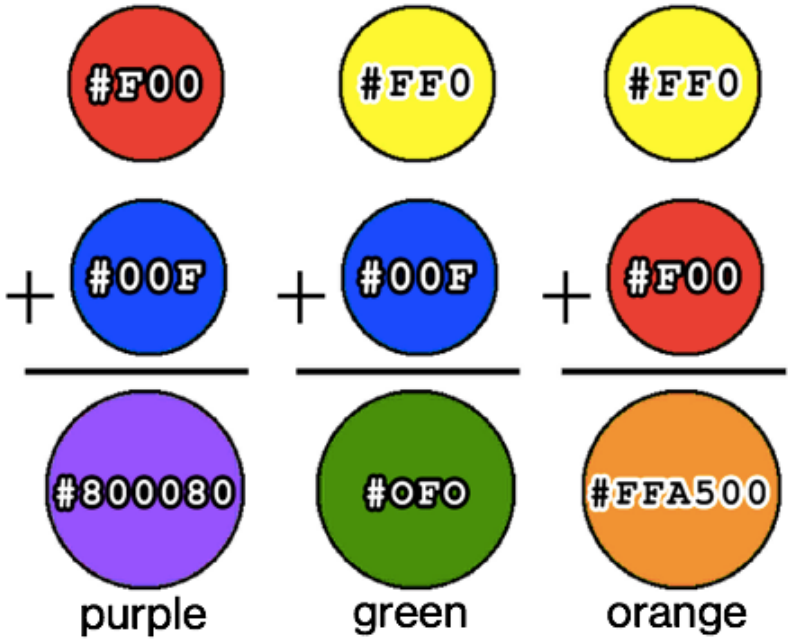
Let's start with some key terms used in this paper:

Given that the system used today is based on the Primary "paint" color mixing, rather than a more modern analogy such as CMYK or even RGB, we must try to remember the "Color Wheel".

ROY G. BIV, as it were.



The important concept to understand is the following additive principle: When you mix together Primary colors, you get Secondary Colors. Red + Blue = Purple. Blue + Yellow = Green. Yellow plus Red = Orange.



Terminology:

Red Team - Offensive security or “ethical hacking” of any type that has been authorized by the organization (penetration testing, physical hacks, black-box testing, compliance testing, social engineering, web app scanning, etc). “The Breakers”

Blue Team - Defensive security, traditionally protection, damage control, and Incident Response (IR). Can also include operational security, threat hunters. Data Forensics (DF). “The Defenders”

Purple Team – Common term for activities combining Red and Blue Teams. These joint activities improve the security posture of a testing scope by building better defenses based on discovered weaknesses. Primary goal is to maximize the results of Red Team activities and improve Blue Team capability.

White Team – All-knowing, neutral, third-party, set the rules of engagement, makes a plan, organizes the other teams, and monitors progress. This could include elements of Compliance, Management, Analysts, and/or logistics (this is where my role mostly operates in the ecosystem). “The Game Masters”

Yellow Team - Individuals who practice the art of creating code, programmers, application developers, software engineers and software architects. “The Builders”. This is an entirely new concept being introduced via this paper.

SDLC – Software Development Lifecycle

Security -

3. Differing Mindsets

Security's Goals:

- Create and implement securely
- Maintain information assets properly
- Test and prove the solution is secure
- Plan for when it can no longer be secure

Security's goals involve secure software development lifecycle (SDLC), post-launch/post-release operational security, compliance, and sunsetting when EOL.

Builder's Goals:

- Time to market
- Correctness
- Optimization
- Minimal defects

Contrast these with the top goals for Builders, which are metrics-based and market-based. There are tradeoffs between Speed vs Cost vs Quality. But shouldn't Quality equate to Security?

4. Poor Interactions

It's no wonder the Builders don't like the Breakers or the Defenders.

We make it far much more difficult for them to reach their goals by imposing tons of requirements on them.

Additionally, Security interaction with Builders is fairly infrequent, and usually only after an audit or after a vulnerability was found in code.

Why does it have to be us vs. them? We are on the same team.

Developers are the first line of defense, but they aren't being given the keys to connect security with their idea of correctness. If software gets released with vulnerabilities that could have been avoided like (XSS), your security program has a gap.

In its paper entitled, "*2015 State of Application Security: Closing the Gap",* SANS explores "the current state of application security through the lens of both builders and defenders". This paper points out several flaws in the current security and builder status

quo. (Ref: https://www.sans.org/reading-room/whitepapers/analyst/2015-state-application-security-closing-gap-35942).

1) Security has a blind spot when it comes to enumerating applications, libraries, etc. The scope is unknown; we don't know what's out there and we can't secure what we don't know about.

2) Everyone is afraid of breaking production, so we test in Staging or Development environments. Unfortunately, these may not always match the production environment *exactly*, and malicious hackers will not care about breaking production (and it's probably the only environment they have access to.

3) We have an inability to sustain iterative RED testing. With Agile and similar methodologies

SANS additionally describes "organizational and communications silos between security, application development and the rest of the organization".

Contrast these flaws with the SANS list of challenges for Builders:

1) Working features need to be delivered on deadline.
2) Lack of management buy-in or funding for security, probably caused by concern over Time to Market aka Time to ROI.
3) Infrequent, negative interactions with infosec
4) Dev and infosec are not on the "same team"
5) Security as "elective" in most degree programs, so Builders lack the skills or knowledge to actually make secure software.

5. "You're Doing It Wrong"

Today, we generally do not push for Yellow to receive security education.

There is an exception to this: We scold them after a Red test, tell them what they did wrong. We, essentially, point out the errors in their art. Imagine that feeling every time they interact with security teams.

I don't like to be wrong. When it comes to security, none of us (including developers) want to do or be wrong in such a way that would cause harm to our coworkers or clients or to our own bonuses. If you didn't care, you wouldn't be here, in this moment, listening this message.

The #dev and #infosec divide of language and goals needs to converge for more secure software

We are heavy consumers of webcasts, podcasts, blogs, twitter, and more, staying on top of the latest security threats and challenges. We are reading "5 ways to {perform (red) I prevent (blue)} XSS attacks", but yellow team is not necessarily reading this in their day-to-day learning. Builder's blogs and periodicals are about form and function, new tricks, and the art of software. Security is sometimes written about, but it's not top priority.

6. Educating Builders

Red Team is considered within the security community to be fun and sexy. We love to hear about software being "broken" by Red Teams. News about vulnerabilities is more exciting than software just being "secure".

Builders are a major component of offensive security testing programs, yet they are often overlooked.

Yellow is building software trying to "maximize correctness".

Let's think about that term, "correctness". It's a very subjective idea – to the developer, to management. Correctness is what we make it. If the security mindset is embedded into the "correctness" frame for the entire Yellow Team, it can become a very powerful force.

Developers need to be regularly exposed to current, live, new and old vulnerabilities, kill chains, architecture decisions, and defensive ideas in order to develop a security mindset.

Requiring encryption or "safe failure" won't explain to a developer how an attack happens or how to protect themselves from it. There is no such thing as completely secure software, or completely bug-free software, so the pragmatic goal is to reduce security bugs, not eliminate them completely.

If insecure code is released to staging or production environments, by the time Red Team finds it, it could be too late.

7. What do the New Teams Do?

We know that the Purple Team helps locate blind spots as well as facilitating improvements in detection and defense, however Purple Team alone fails to address The Big Picture. These are Security Teams working with other Security Teams. We already share the same mindset. It's very efficient and everyone speaks the same jargon, but it doesn't help with what's coming from upstream: the actual software.

All interactions with the Yellow Team are intended to improve software security at the source by educating Builders. The time and effort afforded towards Yellow integration is an investment in software security maturity over time. The "Yellow Builds It, Red Breaks It, Blue Defends It, Yellow Fixes It" model fosters a decrease in security bugs over time when developers are properly educated.

If your corporate mandate is to perform these Team actions, i.e. if your policies say it's part of business as-usual (BAU), then it will happen (because Compliance will say it has to happen!).

7.1. The ORANGE Team

The Orange Team consists of ongoing and/or formally structured interactions between Red and Yellow Team members primarily in order to provide education and benefit to the Yellow team.

One of the Red Team's roles is to share and review findings with Yellow. I don't just mean saying to them, "We recommend that you sanitize user input received by the login page". Why not: Because what's the first thing they are going to do? Search for "how to sanitize user input <XYZ language>". Some code will be tweaked to fix the section where the bug was found, and a non-security-minded developer will probably move on.

Developers can impact defense and reduce risk on a day-to-day-basis, which is why it's so important to take the time to provide them with education about findings, about the design of the application, about whatever it takes. We need to explain to them **HOW** unsanitized user input can be used against the app, and **WHY** sanitization is important. By helping a Builder *REALLY* understand the threat they are trying to avoid, Yellow Team can include security in their personal frame of "correctness".

Orange Team discussions should take place completely outside of Executive reporting, as Orange Teams are extremely "in-the-weeds" and technical.

7.2. The GREEN Team

Blue Team may not always have the data needed to defend or investigate, unless such facilities were added by Yellow Team. Lack of adequate logging or monitoring can lead to worthless investigations or breached systems with no notification.

The Green Team consists of ongoing and/or formally structured interactions between Blue and Yellow Team members. The primary goal of this Secondary Team is to improve code-based and design-based defense (detection, IR, and data forensics (DF) capability created by Yellow Team.

Blue Team provides feedback for Yellow Team, either via gained insight from Purple Team, or threat modeling, giving requirements and discussing solutions for:

- DFIR output
- Log Generation & Activities
- Capability for introspection
    - Reference: http://gauss.ececs.uc.edu/Courses/c6056/pdf/logging.pdf
- Log content/events
- Log generation
    - Something as simple as timezone sync
- Change Management
- Integrity Monitoring
- Anti-V, Anti-M
- Full coverage monitoring

Yellow Team shares limitations of software with Blue Team, so associated risk can be reduced.

Yellow prioritizes and addresses findings with White and/or Compliance, but ONLY after fully understanding them, what causes the issue, what is the threat, and how else it might manifest.

8. How Do We Get From Here to There?

Security management must see Developer and Security teams as part of the same united front, with shared goals and metrics. Metrics must be defined prior to implementing anything you want to measure, or else, how will you know if the effort is worthwhile or successful? If “Security” isn’t a bug count category, it needs to be. Security bug counts per developer, per dev team, per language will likely need to be tracked.

Policies must be put in place to require this type of interaction. External auditors (e.g. PCI, ISO 27001, FISMA) will ask for evidence that proves the Policy statements are being met. This ensures the inter-Team interactions happen, but not that they are productive. It is up to the White Team or Coordinator to ensure exercises involving cross-functional teamwork are occurring as desired.

Misuse Cases are great for iterative SDLC processes (e.g. Agile, Waterfall, etc). When implemented within the Requirements/Use Case definition phase, they will help software architects, designers, and developers understand things the software should NOT be able to do, and reverse their mode of thinking to foster some healthy attack-based paranoia.

9. What's at Stake?

Time, Money, Reputation. No pressure.

If we sacrifice some of our time today training others, intra-Team or inter-Team, we are rewarded with decreased risk, which continues to lessen with time as the non-Security Yellow Team synchronizes with Security; the gap narrows.

If Security shares its vast knowledge with Yellow Team, we can make it past the frustration of seeing the same mistakes repeated, past putting the organization at more risk than it needs to be, and towards a future where software has less security bugs because the Builders understand what they are defending, why, and how they should do it.

Time to Market may be the most important factor to some companies, but we are focusing on teamwork that can build better software over time. Orange and Green teams are part of a maturity model for software, they are not a quick fix. Builder-inclusive teams help narrow the knowledge and experience gap by educating Builders in the Security process and elevating their mindset through increased awareness of how and why software should be built securely.