

Monitoring Attack Surface to Secure DevOps Pipelines

Dan Cornell

[@danielcornell](https://twitter.com/danielcornell)

Agenda

- Background
- Importance of Attack Surface
- What Does Attack Surface Have to Do with DevOps?
- Hybrid Analysis Mapping (HAM) Background
- Installation Instructions
- Use Cases
- Questions

My Background

- Dan Cornell, founder and CTO of Denim Group
- Software developer by background (Java, .NET, etc)
- OWASP San Antonio
- OWASP OpenSAMM Benchmark



Denim Group Background

- Secure software services and products company
 - Builds secure software
 - Helps organizations assess and mitigate risk of in-house developed and third party software
 - Provides classroom training and e-Learning so clients can build software securely
- Software-centric view of application security
 - Application security experts are practicing developers
 - Development pedigree translates to rapport with development managers
 - **Business impact: shorter time-to-fix application vulnerabilities**
- Culture of application security innovation and contribution
 - Develops open source tools to help clients mature their software security programs
 - *Remediation Resource Center, ThreadFix*
 - OWASP national leaders & regular speakers at RSA, SANS, OWASP, ISSA, CSI
 - World class alliance partners accelerate innovation to solve client problems

OWASP ZAP

- Open source web proxy and dynamic application security testing tool
- https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project

Example Codebases

- Bodgelt Store
 - Example vulnerable web application
 - <https://github.com/psiinon/bodgelt>
- Java Spring Petstore
 - Example Spring application
 - <https://github.com/spring-projects/spring-petclinic>
- Railsgoat
 - Example vulnerable web application
 - <https://github.com/OWASP/railsgoat>

ThreadFix Community Edition

- Application vulnerability management
 - And some other stuff
- <https://github.com/denimgroup/threadfix>

Downloads

- <https://dl.dropboxusercontent.com/u/737351/endpoints-json.jar>
- <https://dl.dropboxusercontent.com/u/737351/threadfix-release-2.zip>
- https://github.com/denimgroup/threadfix-examples/tree/master/web_app_attack_surface

Importance of Attack Surface



Importance of Attack Surface

- This is where an attacker can “reach out and touch” your application
 - Web: Mostly in the HTTP request: URL, parameters, headers (cookies)
 - Mobile, IoT: More complicated
 - We will focus on web today
- Target for dynamic testing
 - Automated DAST
 - Manual assessment/penetration testing

What Does Attack Surface Have to Do With DevOps?

- ~~• If you want your talk to be accepted, it has to have DevOps in the title~~
- Let's look at what we want from security in the DevOps pipeline

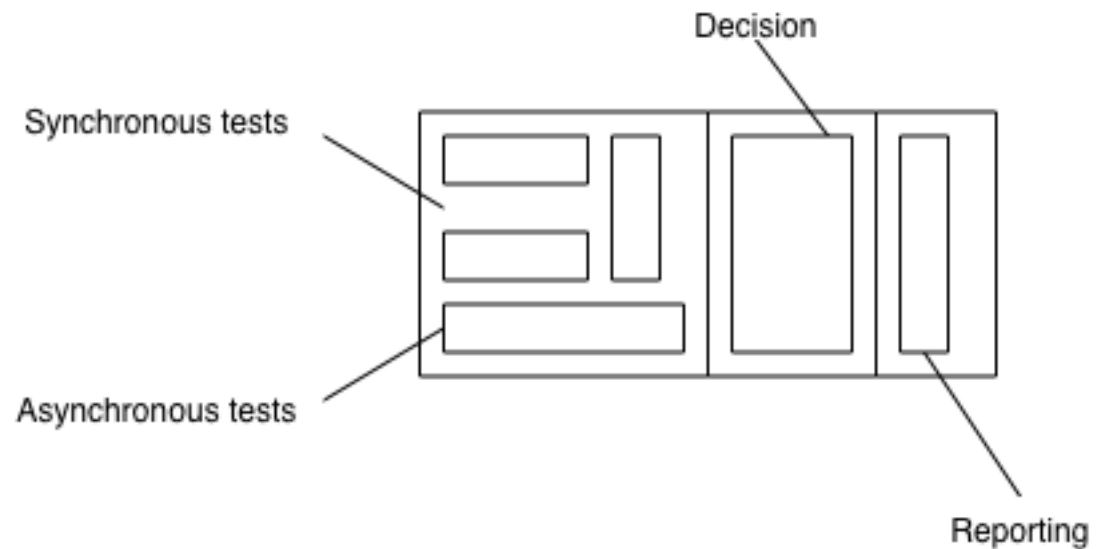
Security in the DevOps Pipeline

Organizations like Etsy and Netflix are doing *amazing* things to secure apps via their DevOps pipelines



Security in the DevOps Pipeline

- Testing
 - Synchronous
 - Asynchronous
- Decision
- Reporting



Focus on Testing in DevOps Pipeline

- Many security tools run too long to include in many pipeline builds
 - Full SAST, DAST
- Security testing also includes manual testing
 - Which is *way* too slow for most pipeline builds
- Tracking attack surface changes over time can help us:
 - Focus testing activities
 - Trigger testing activities

Hybrid Analysis Mapping

- Goal: Merge the results of SAST and DAST testing
- Funded via DHS S&T SBIR contracts
- Facilitated the creation of our attack surface modeling engine

Department of Homeland Security Support

- Currently in Phase 2 of a DHS S&T CSD SBIR
- Acronyms!
 - DHS = Department of Homeland Security
 - S&T = Directorate of Science and Technology
 - CSD = CyberSecurity Division
 - SBIR = Small Business Innovation Research
- Geared toward developing new technologies for Federal customers
- Hybrid Analysis Mapping (HAM)
- Technology has been included with ThreadFix
- Has also resulted in some other released components we will talk about today
- **Please do not assume this talk is endorsed by DHS**
 - **This is just me talking about what we have done**



**Homeland
Security**

Science and Technology

Hybrid Analysis Mapping (HAM)

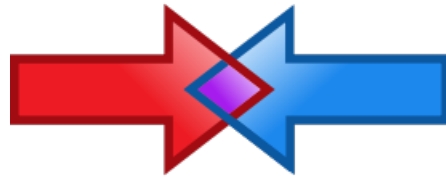
- Initial goal: Correlate and merge results from SAST and DAST
- After we made that work, we found other stuff we could do with the technology

Hybrid Analysis Mapping (HAM)

- Determine the feasibility of developing a system that can reliably and efficiently correlate and merge the results of automated static and dynamic security scans of web applications.



**HP Fortify SCA
Standard**



IBM AppScan

Dynamic Application Security Testing (DAST)

- Spider to enumerate attack surface
 - Crawl the site like Google would
 - But with authentication / session detection
- Fuzz to identify vulnerabilities based on analysis of request/response patterns
 - If you send a SQL control character and get a JDBC error message back, that could indicate a SQL injection vulnerability
- A finding looks like (CWE, relative URL, [entry point])

Static Application Security Testing (SAST)

- Use source or binary to create a model of the application
 - Kind of like a compiler or VM
- Perform analysis to identify vulnerabilities and weaknesses
 - Data flow, control flow, semantic, etc
- A finding looks like (CWE, code/data flow)

```
String username = request.getParameter("username");  
String sql = "SELECT * FROM User WHERE username = '" + username + "'";
```

```
Statement stmt;  
stmt = con.createStatement();  
stmt.execute(sql);
```

Hybrid Analysis Mapping Sub-Goals

- Standardize vulnerability types
 - Settled on MITRE Common Weakness Enumeration (CWE)
- Match dynamic and static locations
 - Use knowledge of language/web framework to build attack surface database
- Improve static parameter parsing
 - Parse out of source code to match with DAST result

Information Used

- Source Code
 - Git, Subversion, Local Copy
- Framework Type
 - Java: JSP, Spring, Struts
 - C#: .NET WebForms, .NET MVC
 - Ruby: Rails
 - PHP: in progress
- Extra information from SAST results (if available)

Unified Endpoint Database

- EndpointQuery
 - dynamicPath
 - staticPath
 - Parameter
 - httpMethod
 - codePoints [List<CodePoint>]
 - informationSourceType
- EndpointDatabase
 - findBestMatch(EndpointQuery query): Endpoint
 - findAllMatches(EndpointQuery query): Set<Endpoint>
 - getFrameworkType(): FrameworkType

Merging SAST and DAST Results

- I have a DAST result:
 - (“Reflected XSS”, /login.jsp, “username” parameter)
- Query the Endpoint Database:
 - Entry point is
com.something.something.LoginController.java, line 62
- Search the other findings for SAST results like:
 - (“Reflected XSS”, source at
com.something.something.LoginController.java, line 62)
- If you find a match – correlate those two findings
- Magic!

That's Great But I Want More

- So our research produced a successful/valuable outcome
 - Hooray
- But – given these data structures, what else can we do?
- From an EndpointDatabase we can:
 - Get *all* of the application's attack surface
 - Map DAST results to a specific line of code
- Given those capabilities we can:
 - Pre-seed scanners with attack surface
 - Map DAST results to lines of code in a developer IDE
 - Map DAST results to lines of code in SonarQube

Scanner Seeding

- What if we could give the DAST spidering process a head start?
- Pre-seed with *all* of the attack surface
 - Landing pages that link in to the application
 - Hidden directories
 - Backdoor or “unused” parameters
- Currently have plugins for OWASP ZAP and BurpSuite
 - Plugin for IBM Rational AppScan Standard is in progress



<https://github.com/denimgroup/threadfix/wiki/Scanner-Plugins>

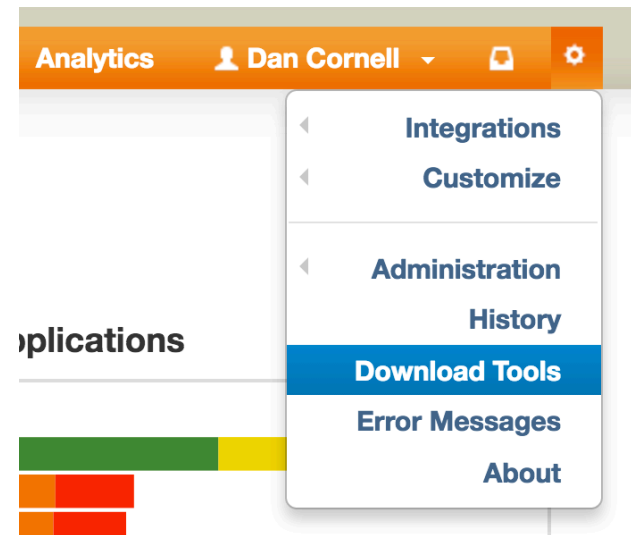
Final Thoughts on SBIR Work with DHS S&T



- Great use of the SBIR program
 - In my humble and *totally* unbiased opinion
- Proved to be the tipping point to developing HAM
 - HAM was *interesting*, but required material investment
- Research produced a successful outcome (we think)
- We found other things we could do with the technology
- Released much of it open source to increase adoption

Getting the Plugin

- Main ThreadFix site
 - <https://github.com/denimgroup/threadfix/>
- ThreadFix build instructions
 - <https://github.com/denimgroup/threadfix/wiki/Development-Environment-Setup>
 - “Running ThreadFix Without an IDE”
- Download plugins from ThreadFix



Plugin Installation Instructions

- OWASP ZAP plugin installation instructions
 - <https://github.com/denimgroup/threadfix/wiki/Zap-Plugin>
- Plugins also available for:
 - Portswigger BurpSuite Professional
 - IBM Rational AppScan (soon)

Attack Surface Enumeration

- Find *all* of the attack surface
 - URLs
 - Parameters that will change application behavior
 - Future: Cookies, other HTTP headers
- Why is this a problem?
 - Hidden landing pages
 - Multi-step processes that automated crawls don't traverse
 - Unknown parameters
 - Debug/backdoor parameters (will discuss this further)
- Great for REST APIs support single-page web applications and mobile applications

Attack Surface Enumeration Benefits

- Reduce false negatives from scanners
 - Better coverage for standard fuzzing
- Pen test *all of* the application

Endpoints CLI Notes

- Syntax: `java -jar [jar-name].jar /path/to/source`
- JAR name will change based on build ID
- After Maven build, can also be found in: `$GIT/threadfix/threadfix-cli-endpoints/target/`
- You want the "-jar-with-dependencies" JAR
- Will output list of HTTP methods, URLs and parameters based on analysis of the source code
- Attack surface!
- Add "-json" to the end of the command to get output in JSON format
 - Easier to manipulate

Command Line Demo

```
target — bash — 121x25
Dans-MacBook-Pro:target dan$ java -jar threadfix-endpoint-cli-2.4-SNAPSHOT-jar-with-dependencies.jar ~/Desktop/DesktopBac
kupWebinar/RiskEUtility/RiskEUtility/
[GET],/AHiddenDirectory/HiddenLaunchPage.aspx,[]
[GET],/AboutRiskEUtility.aspx,[]
[POST GET],/ContactUs.aspx,[txtMessage txtSubject]
[GET],/Default.aspx,[]
[GET],/Home.aspx,[]
[POST GET],/LoginPage.aspx,[txtUsername txtPassword]
[POST GET],/MakePayment.aspx,[txtCardNumber txtAmount]
[POST GET],/Message.aspx,[Msg]
[POST GET],/ViewStatement.aspx,[StatementID]
To enable logging include the -debug argument
Dans-MacBook-Pro:target dan$
```

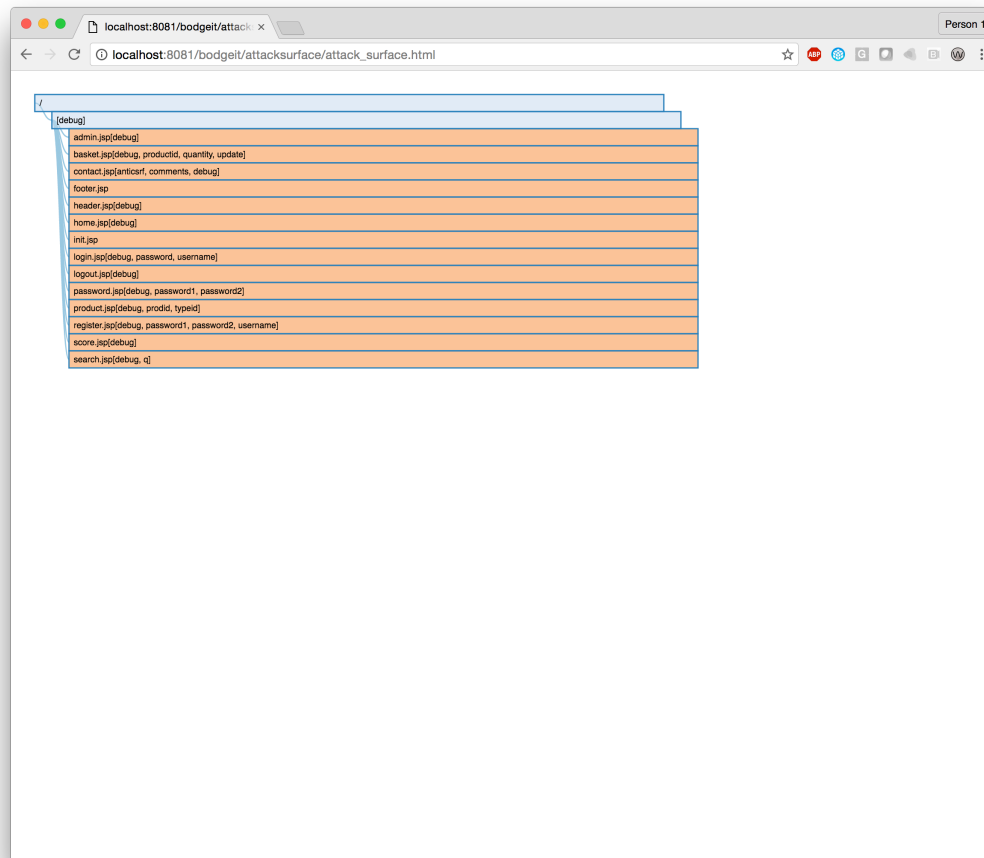
Scanner Attack Surface Seeding Demo

The screenshot displays the Burp Suite Professional v1.6.28 interface. The top menu bar includes 'Burp', 'Intruder', 'Repeater', 'Window', and 'Help'. Below the menu is a toolbar with buttons for 'Target', 'Proxy', 'Spider', 'Scanner', 'Intruder', 'Repeater', 'Sequencer', 'Decoder', 'Comparer', 'Extender', 'Options', 'Alerts', and 'ThreadFix'. The main window is divided into several panes:

- Site map:** Shows a tree view of the target site 'http://localhost:8081'. The 'bodgeit' folder is expanded, revealing various pages like 'about.jsp', 'admin.jsp', 'advanced.jsp', 'basket.jsp', 'contact.jsp', 'footer.jsp', 'header.jsp', 'home.jsp', 'init.jsp', 'js', 'login.jsp', 'logout.jsp', 'password.jsp', 'product.jsp', 'register.jsp', 'score.jsp', and 'search.jsp'. Some items have a magnifying glass icon, indicating they are part of the scanner's attack surface.
- Contents:** A table listing the contents of the selected page. The table has columns for 'Host', 'Method', and 'URL'. The selected row is 'http://localhost:8081 GET /bodgeit/advanced.jsp'. Other rows include various GET and POST requests to different paths under '/bodgeit/'.
- Request/Response:** The 'Request' tab is active, showing the raw HTTP request for 'GET /bodgeit/advanced.jsp'. The request details are: 'HTTP/1.1', 'Host: localhost:8081', 'Accept: */*', 'Accept-Language: en', 'User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)', and 'Connection: close'.
- Issues:** This pane is currently empty.
- Advisory:** This pane is also currently empty.

At the bottom of the interface, there is a search bar with the text 'Type a s' and '0 matches'.

Attack Surface Visualization Demo



Attack Surface Comparison Visualization Demo

The screenshot shows a web browser window with the address bar displaying `localhost:8081/bodgeit/attacksurface/attack_surface_side_by_side.html`. The browser window is titled "Person 1". The main content area is split into two vertical panels, each displaying a list of attack surface items. Each item is represented by a blue header with a dropdown arrow and an orange body containing the item name and its associated parameters.

Left Panel:

- [debug]
- admin.jsp[debug]
- basket.jsp[debug, productid, quantity, update]
- contact.jsp[anticsrf, comments, debug]
- footer.jsp
- header.jsp[debug]
- home.jsp[debug]
- init.jsp
- login.jsp[debug, password, username]
- logout.jsp[debug]
- password.jsp[debug, password1, password2]
- product.jsp[debug, prodid, typeid]
- register.jsp[debug, password1, password2, username]
- score.jsp[debug]
- search.jsp[debug, q]

Right Panel:

- [debug]
- about.jsp[debug]
- admin.jsp[debug]
- advanced.jsp[debug, q]
- basket.jsp[debug, productid, quantity, update]
- contact.jsp[anticsrf, comments, debug]
- footer.jsp
- header.jsp[debug]
- home.jsp[debug]
- login.jsp[debug, password, username]
- logout.jsp[debug]
- password.jsp[debug, password1, password2]
- product.jsp[debug, prodid, typeid]
- register.jsp[debug, password1, password2, username]
- score.jsp[debug]
- search.jsp[debug, q]

Diffing Attack Surface Demo

```
web_app_attack_surface — bash — 80x24
[12]: URL: /score.jsp, Parameters: debug
[13]: URL: /search.jsp, Parameters: q, debug
[0]: URL: /about.jsp, Parameters: debug
[1]: URL: /admin.jsp, Parameters: debug
[2]: URL: /advanced.jsp, Parameters: q, debug
[3]: URL: /basket.jsp, Parameters: quantity, debug, productid, update
[4]: URL: /contact.jsp, Parameters: comments, debug, anticrsf
[5]: URL: /footer.jsp, Parameters:
[6]: URL: /header.jsp, Parameters: debug
[7]: URL: /home.jsp, Parameters: debug
[8]: URL: /login.jsp, Parameters: password, debug, username
[9]: URL: /logout.jsp, Parameters: debug
[10]: URL: /password.jsp, Parameters: debug, password2, password1
[11]: URL: /product.jsp, Parameters: debug, typeid, prodid
[12]: URL: /register.jsp, Parameters: debug, password2, password1, username
[13]: URL: /score.jsp, Parameters: debug
[14]: URL: /search.jsp, Parameters: q, debug
Added attack surface: /about.jsp, /advanced.jsp
Deleted attack surface: /init.jsp
Diff JSON is: {"orig_path_count": 14, "current_path_count": 15, "added": ["/about.j
sp", "/advanced.jsp"], "deleted": ["/init.jsp"]}
Added percent: 0.142857142857
Deleted percent: 0.0714285714286
Dans-MacBook-Pro:web_app_attack_surface dan$
```

Applications for DevOps Pipelines

- Target DAST testing to focus on new attack surface in latest build
 - “Run an authenticated ZAP scan against the three new URLs added in the last commit”
- Set thresholds for when manual assessment/penetration testing is triggered
 - “Schedule a manual penetration test when the attack surface has increased by 10 URLs”
 - “Schedule a manual penetration test when the attack surface has increased by 5%”
 - Focus those efforts on new attack surface
- ChatOps: Attack surface delta notifications on commit
 - “Commit beb78c835706efe5d619148b9a8dc9e35ee9572b added attack surface: /advanced.jsp, /preferenes.jsp”

Next Steps

- Expand the model of application attack surface
 - Currently: Parameters, HTTP verbs
 - Working on: HTTP headers (cookies)
 - Future: Other application types: Mobile, IoT
- Better visualization
 - More details
 - Better granularity
- Native integrations: Jenkins, Slack, HipChat, etc
 - This is kind of “scripty” right now

Questions / Contact Information

Dan Cornell

Principal and CTO

dan@denimgroup.com

Twitter @danielcornell

(844) 572-4400

www.denimgroup.com

www.threadfix.it